Network Working Group
Request for Comments: 3018
Category: Experimental

A. Bogdanov
NKO "ORS"

December 2000

Unified Memory Space Protocol Specification

Status of this Memo

Copyright Notice

Abstract

This document specifies Unified Memory Space Protocol (UMSP), which gives a capability of immediate access to memory of the remote nodes.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [2].

The following syntax specification uses the augmented Backus-Naur Form (ABNF) as described in RFC-2234 [3].

Table of Contents

1 Introduction

UMSP is the network connection-oriented protocol. It corresponds to session and presentation layers of model OSI. The protocol is designed for implementation in a wide class of systems, from simple devices based on the dedicated processors, up to universal computers and clusters.

For the data exchange, the protocol uses transport layer service with reliable delivery. It is possible to use not providing reliable delivery protocol for the transmission of not requiring acknowledgement data. This document describes use TCP and UDP.

The creation of network environment for the organization 128-bit address space of memory distributed between Internet nodes is the basic purpose of the protocol UMSP. The protocol defines algorithm of the connections management and format of network primitives. It doesn't control local memory on the node.

As against the traditional network protocols, the user applications on different nodes interact not by the network primitives exchanging or working with the dataflows, but by immediate data reading/write or control transfers to the code in virtual memory of the remote node. The user's application can know nothing about existence of the protocol and network, and simply use the instructions with 128-bit addresses.

Firstly, it is supposed to use UMSP in systems based on the virtual machines (VM), executing the pseudo-code. However, the protocol may be used in systems executing a processor code, for example, in clusters or in universal operational systems, for the organization of the distributed virtual address space. Besides, the minimal profile of the protocol may be used in simple devices, which do not have the operational system.

The protocol gives various means for set the connection parameters and allows building systems with a high protection level without restriction applications functionalities.

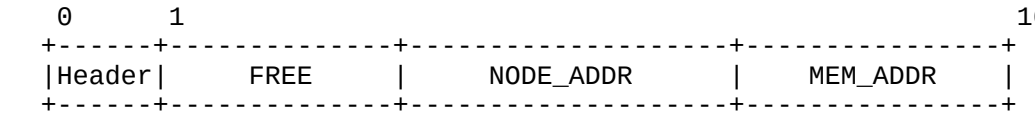
UMSP can essentially simplify the distributed systems development process. It gives an opportunity to unite not only information, but also calculating resources of the large number of polytypic computers without significant expenses for the programs standardization and development.

2 The UMSP Model

2.1 128-bit Address Space

UMSP is based on the 128-bit distributed address memory space model. The 128-bit address contains the information about the network type, network node address and local memory address. It has the following format:

Octets

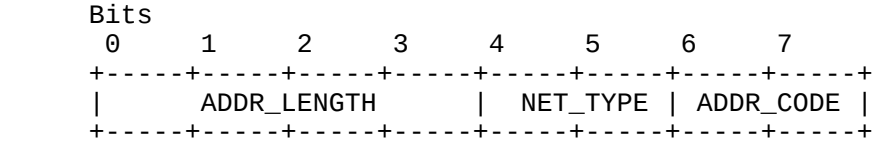


Complete address length is fixed and is equal to 16 octets.

Header

1 octet. Address header field completely defines the address format. The header has the following format:

Bits



ADDR_LENGTH

4 bits. The length of the network address. This field contains the number of octets in the NODE_ADDR field. The value 0 is not allowed.

NET_TYPE

2 bits. The network type. This field specifies a type of network, in which the node is.

ADDR_CODE

2 bits. The length code of the local memory address. The value of this field specifies the length of the local memory address. The following values of the field and appropriated to them length of the field MEM_ADDR are defined:

%b00 - 16 bit
%b01 - 24 bit
%b10 - 32 bit
%b11 - 64 bit

The values combination of the three fields of heading is named address format number. These fields unequivocally define a network, in which the node is located. Format number writes as follows:

N <ADDR_LENGTH> - <NET_TYPE> - <ADDR_CODE>

For example, N 4-0-2 defines the address with length of the node network address 4 octets and memory address with the length 32 bits. The network type 0 for such address format is defined for the network IPv4 in the presented document. If the network type is equal to zero, it may be missed during the writing of the address format number. For example, format N 4-0-2 and 4-2 are equivalent. If both fields NET_TYPE and ADDR_CODE are set to zero, they may be omitted. Thus, a format number writes as one figure.

One or several address format numbers must be assigned for each global network, included in unified system.

FREE

0 - 12 octets. This field is unused by the protocol. It may contain any additional information, which is necessary for the control system of the node memory. If this field is not used, the zero value must be set in all octets. Using of this field results that the network instructions must contain only complete 16 - octet address and the short address of local memory cannot be used.

NODE_ADDR

1 - 13 octets. The node address. The format of this field is defined separately for each address format number. The field of the node address should not necessary precisely correspond to the real network address. If the real network address is longer than this field, it is necessary to organize in the network a subset of supporting the protocol UMSP addresses.

MEM_ADDR

16/24/32/64 bits. The address of local memory. This field is the memory address in system, which is set by a field NODE_ADDR. The node completely responds for its memory control. The protocol does not define the order of using and format of this field.

128-bit address for the user applications is one field. The user code cannot know about a physical arrangement of addressed memory.

The 128-bit memory address may be transmits between nodes, as the data, for example, in the buffer of function parameters, or in the instruction of copying the data. Therefore, it must identify the given node from any other nodes unequivocal.

Any certain algorithm, connecting real network and 128-bit address, does not exist. All used address formats must be known beforehand.

As UMSP has its own address space, it can unite several global networks. The nodes can have internal local networks or subordinated addressable devices connected with the node by the not-network communications. Any node by address format number must have an opportunity to define the gateway respond for routing of this address.

2.2 Computing Model

Computing model is three-layer:

(1) Job
(2) Task
(3) Thread of control

The job corresponds to the user application. The job is distributed and can simultaneously be executed on many nodes. The job control is carried out centralize, from the node named as Job Control Point (JCP). One JCP can control the some jobs. JCP can be located on the same node, on which the job is created, or on any other addressed net point.

The task is the job presentation on the separate node. The task includes one or several computing threads of control. The job has only one task on each node.

The job is finished, when the appropriate user application is finished. At the end of the job all tasks of this job on all nodes are finished.

The job has its isolated 128-bit address space. The address space is segmented. A segment is the local memory of one node. Besides, the protocol allows working with objects. The objects are separate associative memory of the node.

The task thread represents the concrete control thread, which are executed by VM in the certain node. The thread can read and write to any address of 128-bit address space of the job. The control transfer to the address from other (remote) node, results to the creation of the new thread on the remote node. The continuous code segment cannot be distributed on several nodes. In addition, it is impossible to receive continuous memory area distributed on several nodes.

The protocol does not demand to support the different tasks of not-crossed memory space from the separate VM node. The supporting of multi-thread is not also the obligatory requirement.

The 128-bit Global Job Identifier (GJID) is defined by protocol. It is assigned on JCP, which will control the job. All active GJID have the unique values in the unified system at each moment of time.

The job can contain VM code of different types. Different types VM can be situated on one or different nodes. The mechanism of association of different VM types in groups on one node is stipulated, so to the non-uniform code can be executed on one node in a context of one job. The groups are described in details in section 9. VM, incorporated in groups, must work in common memory space (to have a common subsystem of memory control).

2.3 System Architecture

System structure, based on using Virtual Machines, is given in the following figure:

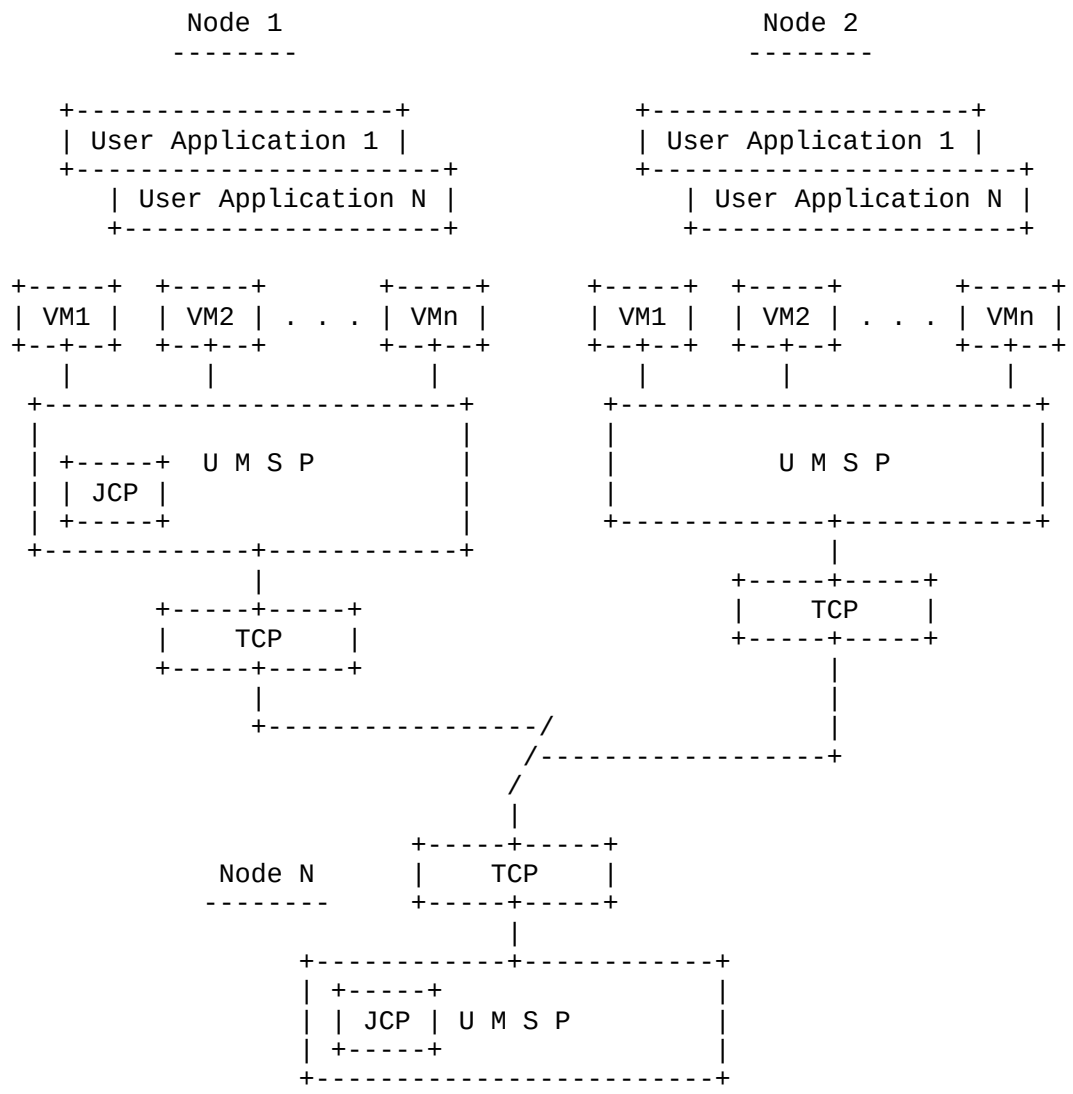


Figure 1. Structure of the system based on use VM.

One or several VM are working on upper level for UMSP. The VM layer is not network level. Last network level is UMSP. Therefore, VM layer has no its own network primitives and uses together with UMSP the same field of operation code.

The end services user of the protocol is the user code, which is executed by the virtual machine. It has the instructions with the 128-bit address. VM translates these instructions to network commands, which are transmitted through the UMSP protocol for the executing by the remote machine. Internal organization VM, command system and API can be anyone. The protocol defines only format of primitives, which the virtual machines exchange through a network.

The protocol does not control the jobs memory. Control of memory should realize VM. If a few VM works on one node, they may have the common memory space or may be completely isolated.

UMSP uses the transport layer with reliable delivery for the data exchange. This document defines of using TCP. For the transfer of not requiring acknowledgement data may be used UDP. Thus, the connection through TCP is obligatory. Use of multiple connections TCP with multiplexing is supposed. The control of transport connections is not the part of the UMSP protocol.

The UMSP instructions do not contain network addresses of the receiver and sender. The protocol requires that one address UMSP must correspond to the one transport layer address. Accordingly, it is necessary to define unequivocal the node address on transport layer by the 128-bit address of memory.

Except the TCP, it is possible to use other transport protocols or not network communications. The following requirements are showed to them:

- Reliable delivery. The transport layer must inform about delivery or its impossibility;
- The violation of a sequence of transmitted segments is allowed;
- The duplication of segments is not allowed;
- At emergency reload of nodes it is necessary to guarantee identification of segments concerning session connections, assigned up to reload;
- Use connectionless-mode is possible.

VM is the independent program and the interaction with the protocol is necessary for it only when it executes the instructions with the 128-bit address, concerning to other node. VM can execute several

user tasks. Each task can contain several threads of control. VM must be able to interpret the application instructions with the 128-bit address to one or several instructions of the UMSP protocol.

The session connection opens between nodes for the data exchange. One connection is relational only with one job. There may be several session connections for the different jobs simultaneously between two nodes. Besides, the protocol provides the connectionless data exchange.

The exchange between UMSP nodes can include the instructions of the following type:

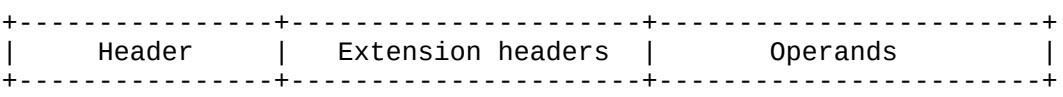
- Immediate reading/write in memory;
- Requests of allocation/free memory;
- Comparison instructions;
- Call-subroutine and unconditional jump instructions;
- Synchronization instructions;
- Work with objects instructions - reading / writing in memory of objects and execution of objects procedures.

UMSP does not trace the user control threads. VM must provide itself the necessary order of performance of the instructions.

The length of UMSP instructions does not depend on segment length of the transport layer. The segmentation is provided for transfer of the long instructions. The packing of the short instructions in one segment with a possibility of compression of headings is used for its transfer. The minimal size of necessary for work segment is 6 octets. For realization of all functions, it is necessary 54 octets.

3 Instruction Format

The UMSP instruction includes the basic header, extension headers and operands. All fields have variable length.



The header contains operation code and the information necessary for the instruction interpretation.

The optional extension headers contain the additional information, not defined in basic header.

The operands contain instructions data.

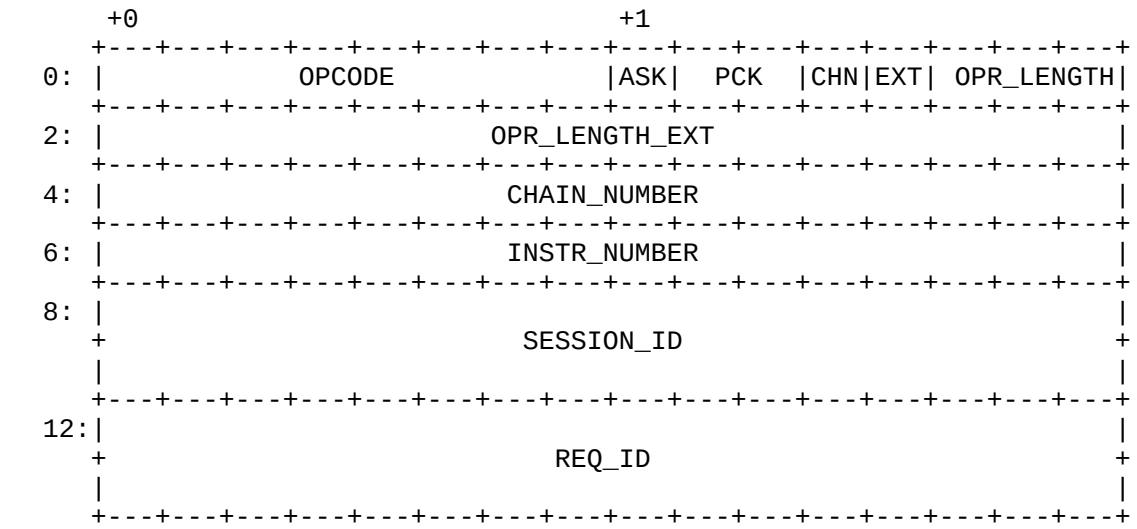
The instruction format allows calculating common instruction length, without knowing definition of separate operation code.

The instructions headers provide for the short and extended format for maintenance of the effective protocol work in wide range of network speeds. Besides, there is a simple algorithm of the headers compression.

The all instructions and extension headers the identifiers are given which enter the name by upper case symbols. The identifiers of the instructions begin with the letter. The identifiers of the extension headers begin with underlining symbol.

3.1 Instruction Header

The header has the following format:



OPCODE

1 octet. The operation code. Value of this field is identified by the instruction. Values of operation codes are divided into the following intervals:

1 - 112	management instructions
113 - 127	reserved
128 - 223	instructions of exchange between VM
0, 224, 255	reserved

ASK

1 bit. The flag of response necessity. This flag defines presence of field REQ_ID in header. If ASK = 1, there is field REQ_ID in the instruction. If EXT = 0, the field REQ_ID in the instruction are absent.

PCK

2 bits. The Header compression attribute. These bits are used for packing instructions headers transmitted on one connection TCP or for sending of the several instructions in one package UDP. Use of these bits is based on the assumption that two following in succession instructions concern to one session connection, or one chain, with a high probability. The PCK bits have one of the following values:

%b00 - The instruction does not belong to the definite session. The fields CHAIN_NUMBER, INSTR_NUMBER and SESSION_ID are absent in header of such instruction.
%b01 - The given instruction concerns to the same session connection, as previous. The field SESSION_ID in the instruction header is absent.
%b10 - The given instruction belongs to the same connection and same chain, as previous. The fields CHAIN_NUMBER, INSTR_NUMBER and SESSION_ID in header of such instruction are absent. The INSTR_NUMBER value of the current instruction calculates by addition of one to INSTR_NUMBER value of the previous instruction.
%b11 - The given instruction may does not concern to the same session, as previous. The field SESSION_ID is present at it. The presence of fields CHAIN_NUMBER and INSTR_NUMBER is defined by CHN flag.

CHN

1 bit. The flag of chain. Transmitted on one session connection and concerning one job instructions, may be unified in a chain. Chains are considered in details by section 7. If SEQ = 1, the instruction is connected with chain and there are fields CHAIN_NUMBER and INSTR_NUMBER (if PCK is not set to %b10) at it. If bit CHN = 0, the instruction is not connected with chains and there are no fields CHAIN_NUMBER and INSTR_NUMBER in it.

EXT

1 bit. The flag of extension headers presence in the instruction. If EXT = 1, there is one or more extension headers in the instruction. If EXT = 0, the extension headers in the instruction are absent.

OPR_LENGTH

3 bits. The number of 32 bit words in the operands field. The value 0 defines absence of operands field. The value %b111 specifies use of the extended header format. In the extended format, the length of operands is defined by the field OPR_LENGTH_EXT, and the field OPR_LENGTH is not used.

OPR_LENGTH_EXT

2 octets. The number of 32 bit words in the operands field. The field OPR_LENGTH_EXT is present in header, only if OPR_LENGTH = %b111. If OPR_LENGTH < > %b111, the field OPR_LENGTH_EXT is absent. If OPR_LENGTH_EXT = 0, the field of operands is absent. There are following reasons, on which it is necessary to use field OPR_LENGTH_EXT instead of OPR_LENGTH:

(1) If operands length must be more than 24 octets
(2) If making the fields alignment of 4 octets is more effective, than compression of header of 2 octets.

CHAIN_NUMBER

2 octets. The number of chain. This field contains number of chain, to which the given instruction concerns. The values %x0000 and %xFFFF are reserved.

INSTR_NUMBER

2 octets. The instruction number. This field contains the serial number of instruction in a chain. The numbering begins with zero. Value %xFFFF is reserved.

SESSION_ID

4 octets. It is the identifier of the session connection assigned by the instruction receiver. During the session connection opening, each side sets its own identifier to connection and informs it to other side. The zero value of this field specifies that the instruction does not concern to the definite session. The value %xFFFFFFFF is reserved.

REQ_ID

4 octets. The request identifier. It is uses for establishment of correspondence between requests and responds to it.

Further, the identifier OPR_LENGTH is used at the description of the instructions format. It means using of OPR_LENGTH_EXT field, if OPR_LENGTH = %b111. The instruction with length of operands, which are not exceeding 24 octets, may be transmitted with header in the short format (OPR_LENGTH < > %b111) or in the extended format (OPR_LENGTH = %b111). Both forms are equivalent.

Minimal header length in the short format is 2 octets, in the extended format - 4 octets. Maximal header length is 16 octets.

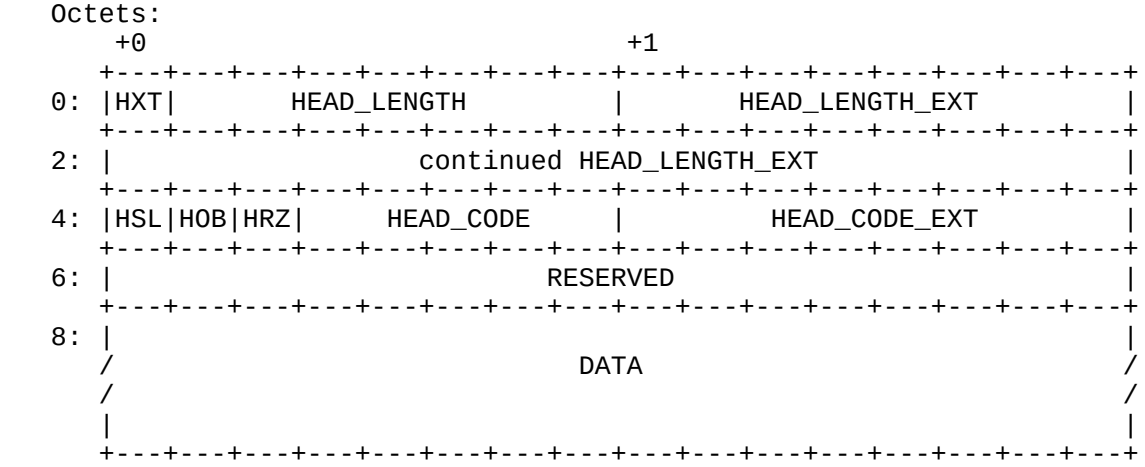
3.2 Extension Headers

If the EXT flag in the instruction header set to 1, the instruction contains from one up to thirty extension headers. The extension headers are used for the following purposes:

- For sending of the service information which were not provided in the basic header.
- For sending of the data of length more than 262240 octets in one instruction.

The extension headers have the following common format:

Octets:



HXT

1 bit. Specify length of the field of data length. If HXT = 0, length of the extension header is defined by a field HEAD_LENGTH. The field HEAD_LENGTH_EXT in this case is absent. If HXT = 1, length of header is defined by unification of fields HEAD_LENGTH and HEAD_LENGTH_EXT.

HEAD_LENGTH

7 bit. The number of 16 bit words in DATA field. If HXT = 0, this is independent field. If HXT = 1, it is the senior bits of complete length field.

HEAD_LENGTH_EXT

3 octets. The number of 16 bit words in DATA field. If HXT = 0, this field is absent. If HXT = 1, it is the younger bits of complete length field.

HSL

1 bit. The flag of last header. It is set to 1 for last extension header in the instruction. In other extension headers, this flag is set to 0.

HOB

1 bit. The flag of obligatory processing. It defines the order of the instruction processing, if the receiving node does not know purpose of the extension header or cannot process it by any reason. If HOB = 1, instruction must not be carried out. If HOB = 0, it does not influence on the instruction processing. The protocol must process all extension headers, irrespective of errors presence.

HRZ

1 bit. The field is reserved for the future expansions. This field must not be analyzed by the protocol on receiving. It must be set to 0 at sending.

HEAD_CODE

5 bits. If HXT = 0, the field contains the extension header code. If HXT = 1, this field joins the field HEAD_CODE_EXT. It is the senior bits of the header code.

HEAD_CODE_EXT

1 octet. If HXT = 0, this field is absent. If HXT = 1, it is the younger bits of the header code.

RESERVED

2 octets. If HXT = 0, this field is absent. If HXT = 1, this field is reserved for further use. The field RESERVED must not be analyzed by the protocol during the receiving in the current realization of the protocol. It must be set to 0 at sending.

DATA

The data field of the extension header. If HXT = 0, the length of field is 0 - 254 octets, if HXT = 1, the length is 0 - 4 * 10^9 octets. The format of this field is defined separately for each value of the header code.

On the receiving side, the extension headers must be processed in that order, in what they follow in the instruction. If the instruction contains more than 30 extension headers, it is considered erroneous. It is necessary to break off the session connection, on which it was transmitted, after the reception of such instruction.

The identifiers HEAD_LENGTH and HEAD_CODE are used further in the text at the description of the extended headers format. It assumes using of fields HEAD_LENGTH + HEAD_LENGTH_EXT and HEAD_CODE + HEAD_CODE_EXT, if HXT = 1. The headers with the code 0 - 30 can be sent in short (HXT = 0) and in extended (HXT = 1) format.

3.3 Instruction Operands

The operands field contains the instruction data. The length of operands field is showed in OPR_LENGTH or OPR_LENGTH_EXT and it is multiple to four octets. If necessary, 1 - 3 zero-value octets are padded in the end of a field. Maximal length of operands is 262140 octets. The extension headers are used, if the instruction must contain longer data.

The format of the operands field is defined separately for each instruction.

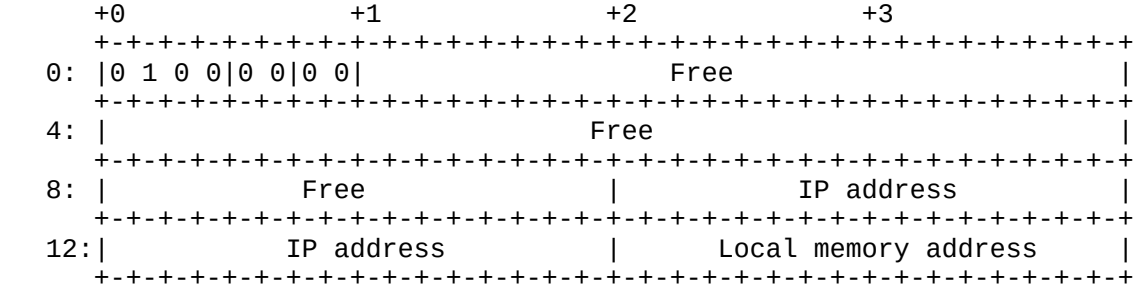
3.4 Address Formats

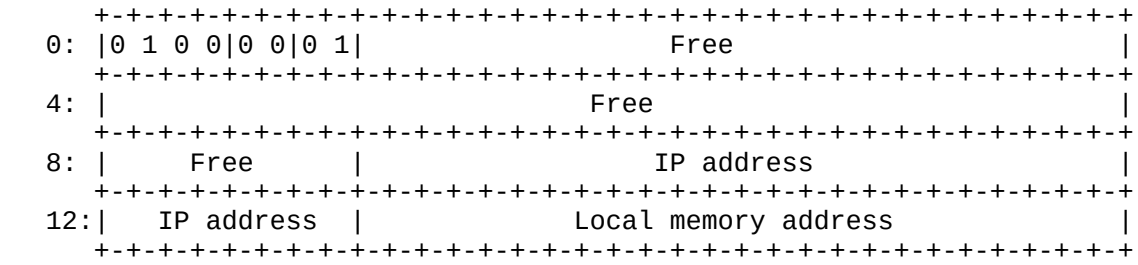
The following address format numbers are definite for nodes, immediately connected to the global IPv4 network:

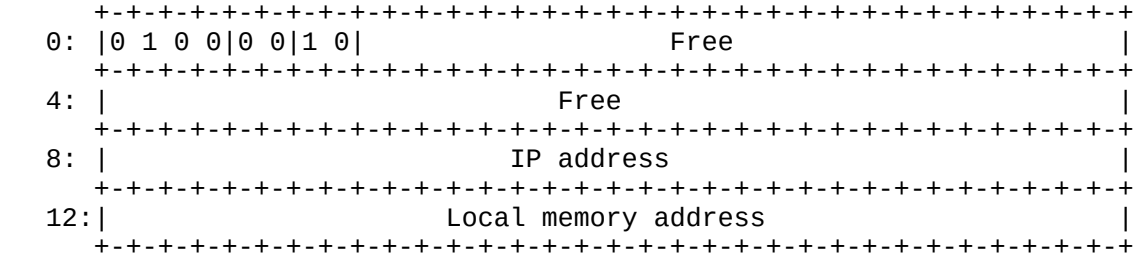
N 4-0-0 (4)
N 4-0-1 (4-1)
N 4-0-2 (4-2)

The appropriate formats of 128-bit addresses:

Octets:







Free

It is not used by the protocol.

IP address

It sets the node address in the global IPv4 network.

Local memory address

It is described in section 2.1.

IP-address defines the nodes of the given type unequivocally. The TCP is used for the interaction with such nodes. For sending of not requiring response instructions, using UDP is allowed. IANA has assigned ports TCP and UDP 2110. This port must be open for the listening (receiving). TCP node, initialing the connection opening, or the UDP node, carrying out the package sending, can use any port. Using several TCP connections with multiplexing is supposed.

4 Response of the Instructions

The protocol instructions are divided into two types:

(1) The management instructions transmitted on UMSP layer (OPCODE = 1 - 112).
(2) The instructions of the exchange between VM (OPCODE = 128 - 223).

The processing of two types of the instructions differs as follows:

- The field of the identifier of request REQ_ID is formed by the protocol in the instructions of the first type, and it is formed by VM for the instructions of the second type.
- The protocol must analyze the field REQ_ID and compare it with the instructions, transmitted earlier, after receiving of the response instruction of the first type.
- The protocol must not analyze the field REQ_ID after receiving of the response instruction of the second type. This instruction is simply sent to VM.

The response instructions have the field ASK equal to 1. It means, that the header have the field REQ_ID. The value taken from the confirmed instruction is written into the field REQ_ID. The response instruction does not require response.

A few VM can be connected to the protocol on the node. Everyone VM can work in its own address space. The identifiers of requests for different VM can coincide. Therefore, instruction is identified by two fields:

- The session identifier SESSION_ID, which is connected with definite VM.
- The request identifier REQ_ID.

4.1 RSP, RSP_P

"Response" (RSP) and "Response of the protocol" (RSP_P) instructions have the identical format. The difference is only in the operation code:

```
OPCODE = 129/1  ; correspondingly to RSP/RSP_P
ASK = 1
PCK = %b01/11
EXT = 0/1
CHN = 0
OPR_LENGTH = 0/1
SESSION_ID and REQ_ID - The values is taken from the confirmed
                        instruction.
Operands:
   2 octets: The basic return code.
   2 octets: The additional return code.
The optional extension header:
   _MSG - contains the arbitrary error description.
```

The instruction without operands is used for the positive response. It is equivalent to zero values of the field of the basic and additional return codes.

The zero basic return code is used for positive response. The additional return code may have non-zero value.

The instruction with non-zero basic return code is used for negative response. The basic return code defines the error category. The additional return code identifies an error.

The instruction RSP is formed upon the VM request. The return codes must be received from VM. If the protocol cannot deliver the requiring response instruction to VM, it forms negative response RSP independently.

The instruction RSP_P is always formed at the UMSP layer. If the protocol cannot define on what instruction the RSP_P is transmitted, nothing actions is executed.

4.2 SND_CANCEL

There can be a necessity to cancel sending after the part of the data have been already transmitted and have occupied the buffer on the reception side, by sending of the long fragmented instructions or transactions. The protocol provides the instruction "The sending is canceled" (SND_CANCEL) for this purpose. This instruction has the following fields value:

```
OPCODE = 2
ASK = 0
PCK = %b01/10/11
EXT = 0/1
CHN = 1
OPR_LENGTH = 1
SESSION_ID - The value is taken from the cancelled chain.
CHAIN_NUMBER - Number of the chain, which sending is cancelled.
INSTR_NUMBER - Always has zero-value.
Operands:
   2 octets: The basic return code.
   2 octets: The additional return code.
The optional extension header:
   _MSG - contains the arbitrary error description.
```

The instruction SND_CANCEL is used for the cancel of the partially transmitted transaction or fragmented instruction. At the receiving the SND_CANCEL instruction, all the earlier received data in the chain are rejected.

5 Jobs Management

The jobs management includes the following functions:

- Initiation and completion of jobs;
- Initiation and completion of tasks;
- Opening and closing of session connections;
- Activity control of nodes.

The instructions with OPCODE = 1 - 112 are used for jobs management. These instructions must be sent through TCP. Use UDP is not allowed, even if the instructions do not demand response.

UMSP bases on model with the centralized control of the separate job. The reason is that the pointers control is not obviously possible in the decentralized system. Any task can be finished at any moment or the node can be reloaded. There is no way guaranteeing the notification about in the decentralized system all other nodes, on which the job works. As the job continues to exist - the task concerning the job can be initiated on the same node again. This task can allocate new dynamic resources. The addresses for the again allocated resources can be crossed with addresses of resources, which existed on the node before the task restart. The old pointers can be kept on other nodes. It may be the formally correct pointers, but they will actually specify other objects. The uncontrollable work of the application can be consequence of such situation.

UMSP solves this task as follows:

- It allows defining the node, on which the task was completed, precisely.
- If the task on the node is finished before end of the job, all nodes, on which the job is executed, are notified of it.
- The repeated task initialization on the node is allowed, while all nodes will receive the message about the first task end.

The protocol does not control the pointers. VM supervises the pointers correctness. VM must have architecture, in which 128 - bit pointers are stored in special memory areas, for this purpose. The protocol informs VM about the nodes, on which task have finished the work. VM must make all pointers concerning such tasks, invalid. It results in exclusive situations at the access under these pointers. If the application provides processing exceptions, it keeps the capacity for work, or it is finished emergency. Such decision allows excluding unguided applications working.

For the decision of the specified questions at UMSP level, the control job node is defined for each job. It names Job Control Point (JCP). It may be the same node, on which the job is initiated, or it can be another dedicated node. The basic JCP function is to trace the initialization and the end of the job tasks. Besides, the dedicated JCP node may be used for the centralized users identification and the attack protection.

The following identifiers are definite for the jobs and tasks control:

- Locally Task Identifier (LTID) is assigned to each active task on the node. LTID length is equal to the length of local memory address defined for the node. All LTID on the node must give unique values at each moment of time. It is allowed to establish LTID, used earlier in the already completed tasks, for the again initiated tasks.
- JCP assigned the Control Task Identifier (CTID) to each task of the job. Its length is equal to length of the local address memory on the node JCP. All CTID on the JCP must give unique values at each moment of time. As against LTID, the CTID value is chosen with some restrictions.
- Globally Task Identifier (GTID) is assigned to each task. GTID has the same format, as the 128 - bit address of node memory has. The address of local memory is replaced on LTID in it.
- Globally Job Identifier (GJID) is assigned to the each job. GJID is defined on the JCP node. It has the same format, as the 128 - bit address of node JCP memory has. The address of local memory

is replaced on CTID of the first (initial) task of the job in it. GJID is used in the procedure of session connection opening for the definition JCP, which controls the job.

LTID and CTID are written at the instructions in the field of length 2/4/8 octets. If the allocated for identifier field in the instruction is longer than identifier, LTID (CTID) writes in the last octets. In the initial octets, the value 0 must be written. If received LTID (CTID) is shorter than the local memory address, it is necessary to pad it with the zero octets in the beginning.

GTID and GJID are written at the instructions in the field of length 4-16 octets. The field FREE is not present at these identifiers (see section 2.1). It is considered, that it contains the zero-value octets. Length of the identifier is defined in header of the address.

By sending of instructions CONTROL_REQ, TASK_REG and SESSION_OPEN, the protocol uses timeout. The value of timeout is assigned by node and must be more than three intervals of the maximal time of delivery at the transport layer. The timeout is not influenced the waiting period in queue to the transport layer.

5.1 Job Initiate

The job concerns to the user application executed on VM. The UMSP job initialization can be made simultaneously with the application user start or during its working.

The task, appropriated to its job, is initialized on the node together with the job. LTID is binding to this task.

If the node, on which the user application was loaded, is chosen for JCP, the question of the job initialization lays beyond the scope of the network protocol.

Other node can be chosen as JCP for the following reasons:

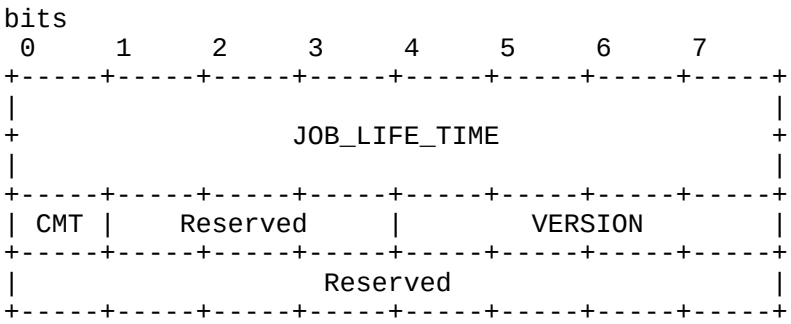
- The job initialization node is connected to network by slow-speed or overloaded channel. It is undesirable to send the managing traffic.
- The node has no computing possibilities for conducting the managing tables.
- The authentication on the detailed node is necessary.

If the other node is chosen for JCP, the node, that initiates the job, must register the job at JCP.

5.1.1 CONTROL_REQ

The instruction "To request a control" (CONTROL_REQ) is sending from the node, initial the job, to JCP of other node. The instruction has the following values of fields:

OPCODE = 3
PCK = %b00
CHN = 0
ASK = 1
EXT = 0/1
OPR_LENGTH = 2/3 ; Depends on LTID length.
REQ_ID - The value is assigned by the sender node protocol and then will be sent in the response.
Operands:
4 octets: The control parameters profile. This field has the following format:



JOB_LIFE_TIME

2 octets. The job lifetime in seconds. The zero-value signifies that the restriction of the job lifetime is unused.

CMT

1 bit. The flag of several JCP using. This field is reserved for the future expansion of the protocol.

VERSION

1 octet. The number of the UMSP version. It must contain the value 1.

Reserved

3 + 8 bits. All bits must be set to 0.

4/8 octet: LTID of task of the job, assigned on the node, which initiate the job (by the sender of this instruction).

The optional extension headers:

_JOB_NAME - This header contains the name of the Job. Is assigned once and must not change further.

_INACT_TIME - This header contains the inaction time (see section 5.7).

At reception of the CONTROL_REQ instruction JCP checks the LTID value from the received instruction and makes the following:

(1) If the node, which has sent CONTROL_REQ, already has registered on JCP the active job with such LTID, the notification about abnormality end of the registered job is sent, as is described in section 5.5.2 (it is considered, that the node was reloaded). After that, the sanction to an initiation of the new job is sent.
(2) If the node has no registered job with received LTID, it allows the new job initiation at once.

If JCP confirms the control, it will send the instruction CONTROL_CONFIRM, or else CONTROL_REJECT.

5.1.2 CONTROL_CONFIRM

The instruction "To confirm the control" (CONTROL_CONFIRM) is sent from JCP as the positive response to CONTROL_REQ instruction. CONTROL_CONFIRM has the following values of fields:

OPCODE = 4
PCK = %b00
CHN = 0
ASK = 1 ; The instruction does not need to be responded. This flag specifies presence of the REQ_ID field.
EXT = 0/1
OPR_LENGTH = 1-4 ; Depends of length of the GJID.
REQ_ID - The value is taken from the instruction CONTROL_REQ
Operands:
4-16 octets: The GJID assigned to the job on the JCP.

The sending of the instruction CONTROL_REQ means request of control and request of task initiation. Assigned to the task CTID is part GJID (field of the local memory address).

5.1.3 CONTROL_REJECT

The instruction "To reject the control" (CONTROL_REJECT) is sent from JCP as the negative response to CONTROL_REQ instruction. CONTROL_REJECT has the following values of fields:

```
OPCODE = 4
PCK = %b00
CHN = 0
ASK = 1.  The instruction does not need to be responded.  This flag
          specifies presence of the REQ_ID field.
EXT = 0/1
OPR_LENGTH = 1/2 ; Depends on presence of the control parameters
                   profile field.
REQ_ID - The value is taken from the instruction CONTROL_REQ
Operands:
   2 octets: The basic error code.  The zero-value is not
             available.
   2 octets: The additional error code.
   4 octets: The control parameters profile (see section 5.1.1),
             that is allowed by JCP.  This is optional field.
The optional extension headers:
   _INACT_TIME - This header contains the inaction time (see
                 section 5.7).
   _MSG - contains the arbitrary error description.
```

5.2 Task Initiate

The job is executed on several nodes simultaneously. The task, appropriate to it, must be initialized on each node. There is corresponding only one task to one job on the node. Each task must be connected only with one job.

The task is initiated together with the job on the node, which had created the job. On the other nodes, the task is initiated during the receiving of the first request on the opening of the session connection, which is appropriate to the job. The request about openings of session connection contains GJID. GJID contains the JCP address. It is necessary to receive the sanction from JCP for the task start. If the request about the opening of session has been received from JCP node, it is not necessary to request the sanction.

5.2.1 TASK_REG

The instruction "To register a task" (TASK_REG) is sent from the node, which initials the task, to JCP of the remote node. The instruction has the following values of fields:

```
      OPCODE = 6/7/8 ; For length CTID of 2/4/8 octets.
      PCK = %b00
      CHN = 0
      ASK = 1
      EXT = 0/1
      OPR_LENGTH = 2-8  ; Depends on length of the GTID and LTID.
      REQ_ID - The value is assigned by the sender node protocol and
               then will be sent in the response.
      Operands:
         2/4/8 octets: CTID of the task initiated the job.  It CTID is a
                       part GJID from the instruction SESSION_OPEN.
         4-16 octets: GTID, assigned on the node, initialed session
                      connection.  GTID is formed of sender addresses (at
                      transport layer) and field LTID of the instruction
                      SESSION_OPEN.
         2/4/8 octets:  LTID, assigned on the node, initialed the task
                        (by the sender of this instruction).
      The optional extension headers:
         _INACT_TIME - This header contains the inaction time (see
                       section 5.7).
```

The instruction TASK_REG must be sent only if the task with given GJID was not initiated on the node.

JCP confirms initiation of a task at observance of the following conditions:

(1) Task with received GTID already has registered on JCP.
(2) Task with LTID for the node requesting for initiation has not registered.

In all other cases, JCP will not confirm a task.

If JCP confirms the task, it will send the instruction TASK_CONFIRM, differently TASK_REJECT.

5.2.2 TASK_CONFIRM

The instruction "To confirm the task" (TASK_CONFIRM) is sent from JCP as the positive response to TASK_REG. TASK_CONFIRM has the following values of fields:

```
      OPCODE = 9
      PCK = %b00
      CHN = 0
      ASK = 1.  The instruction does not need to be responded.  This flag
                specifies the field REQ_ID presence.
      EXT = 0/1
```

```
      OPR_LENGTH = 1/2  ; Depends on length of the CTID.
      REQ_ID - The value is taken from the instruction TASK_REG.
      Operands:
         4/8 octets: The CTID assigned to the task on the JCP.
      The optional extension headers:
         _JOB_NAME - This header contains the name of the Job.
```

5.2.3 TASK_REJECT

The instruction "To reject the task" (TASK_REJECT) is sent from JCP as the negative response to TASK_REG instruction. TASK_REJECT has the following values of fields:

```
      OPCODE = 10
      PCK = %b00
      CHN = 0
      ASK = 1.  The instruction does not need to be responded.  This flag
                specifies presence of the REQ_ID field.
      EXT = 0/1
      OPR_LENGTH = 1
      REQ_ID - The value is taken from the instruction CONTROL_REQ
      Operands:
         2 octets: The basic error code.  The zero-value is not
                   available.
         2 octets: The additional error code.
      The optional extension headers:
         _INACT_TIME - This header contains the inaction time (see
                       section 5.7).
         _MSG - contains the arbitrary error description.
```

5.2.4 TASK_CHK

With the purposes of a safety the node, which have received request about the opening of session connection, may check up at JCP the node, which has initialed connection, even if the task was already initiated.

The instruction "To check up the task" (TASK_CHK) is sent from the node, which has received the instruction of the establishment of session connection SESSION_OPEN, to JCP. The task with given GJID, must have existed on the node already. The instruction TASK_CHK format coincides with TASK_REG. OPCODE = 11. The response to the instruction TASK_CHK JCP forms instructions TASK_REG similarly.

JCP confirms the instruction TASK_CHK if a task with received GTID and LTID already has registered on JCP.

The sending of the TASK_CHK is optional.

5.3 Establishment of session connection

The session connection is established between two tasks of one job. The connection is established under the VM initiative and it is used for the exchange of the instructions between VM.

One session connection must be connected only with one task on the node. The task may have several connections with different nodes. Between two nodes must be only one session connection with one GJID.

The request about the establishment of session connection contains the global identifier of the job GJID. If the node receives the request about the establishment of connection with GJID, which is not presented on the given node, VM must create a new task. If the task has been already initialized, the new task is not created.

The session connection needs to be established over TCP. After the connection is established, the sending of the instructions, which are not require of execution response, is possible through UDP. One TCP connection may be used by several session connections. One session connection may use several TCP connections.

The protocol allows working without the establishment of session connection. The node must have VM by default, which must execute the instructions without the establishment of connection.

At the establishment of session connection, the sides agree about the used VM type and the subset of the protocol functions. The session connection UMSP may be asymmetrical. It means, that two sides of one connection can be connected with VM of the different type and provide the different subset of the protocol functions.

If at an establishment of session connection the zero-type VM is used, it specifies group VM (see section 9). The zero-value of realization VM is not allowed.

The procedure of the establishment of session connection may contain from 2-way up to 8-way handshakes.

5.3.1 SESSION_OPEN

The instruction "To open a session" (SESSION_OPEN) is used for the initiation of session connection and for the specification of connection parameters during handshake. It has the following values of fields:

```
      OPCODE = 12
      PCK = %b00/11.  In the first instruction (initial) the value of
                       this field is set to %b00.  In all subsequent -
                       %b11.
      CHN = 0
      ASK = 1
      EXT = 0/1
      OPR_LENGTH = 6 - 10 ; Depends on length GJID and LTID.
      SESSION_ID - In the first instruction this field is absent.  In all
                   subsequent, it contains the identifier of sessions,
                   assigned by the instruction receiver.
      REQ_ID - This field contains the session connection identifier,
               assigned by the instruction sender.
      Operands:
         2 octets: The VM type required from the addressee.
         2 octets: The VM version required from the addressee.
         4 octets: The profile of connection required from the
                   instruction addressee.
         2 octets: The VM type of the sender.
         2 octets: The VM version of the sender.
         4 octets: The profile of connection given by the instruction
                   sender.
         2 octets: The number of 256 octet blocks in the buffer,
                   allocated for session ("window"), on the side of the
                   sender of this instruction (see section 7.4).  The
                   zero-value specifies absence of the buffer.
         4-16 octets: GJID.
         4/8 octets: LTID of the sender task, assigned on the node -
                     sender of the instruction.  It is used in the
                     instruction TASK_REG (as a part of the field GTID).
```

If the VM type and version, required from the addressee, have the value 0, the receiving node independently chooses the VM type and reports it in the response. The establishment of connection without binding to VM or VM group is not allowed.

Totally, it can be transmitted up to 7 instructions SESSION_OPEN at the establishment of connection. The instruction SESSION_ACCEPT is used for the response of the establishment of connection. For the refusal of connection the instruction, SESSION_REJECT is used.

It is possible to refuse connection on any step. It is necessary either to confirm connections, or to refuse it on the eighth step.

During the establishment of connection the following parameters may be changed:

- o VM type and VM version;
- o profiles of connection.

If the repeated request about opening of session connection is received from the definite node, while one connection with received GJID have been already established, the following variants are possible:

(1) If the request has arrived from the node JCP, it is necessary:
    - o To finish the existing task emergency and to deallocate all dynamic resources belong to it.
    - o To initiates a task without request of the JCP sanction again.
    - o To confirm the establishment of connection.

(2) If the request arrived not from the JCP node, it is necessary to refuse the establishment of new session connection. The existing task does not need to be changed.

5.3.2 SESSION_ACCEPT

The instruction "To accept the session" (SESSION_ACCEPT) is used for positive response to the establishment of session connection. It has the following values of fields:

```
OPCODE = 13
ASK = 1
PCK = %11
EXT = 0/1
CHN = 0
OPR_LENGTH = 0
SESSION_ID - This field contains the session connection identifier
             of assigned by the node of the addressee of the
             instruction.
REQ_ID - This field contains the session connection identifier,
         assigned by the instruction sender.
```

5.3.3 SESSION_REJECT

The instruction "To reject the session" (SESSION_ACCEPT) is used for negative response to the establishment of session connection. It has the following values of fields:

```
OPCODE = 14
ASK = 0
PCK = %b11
EXT = 0/1
CHN = 0
OPR_LENGTH = 1
```

SESSION_ID - This field contains the session connection identifier of assigned by the node of the addressee of the instruction.
Operands:
2 octets: The basic error code. The zero-value is not available.
2 octets: The additional error code.
The optional extension headers:
_MSG - contains the arbitrary error description.

5.3.4 Connection Profile

The profile of connection is defined in 4-octet field of flags. The flags have identifiers S0 - S31. The number in the identifier is defining the serial number of bit. If the flag is set to 1, the function, connected with it, is provided. If the flag is set to 0, the function, connected with it, is not provided (not required). The list of functions, determined at the establishment of session connection, are described further.

Work with chains:

S0 - Use of fragmented instructions.
S1 - Use of sequences.
S2 - Use of transactions.

Establishment of connection:

S3 - Use the exchange of the data without the establishment of connection.
S4 - Use the exchange of the data with the establishment of connection.

The instructions format:

S5 - Reserved. Must have set to 0.
S6 - Use of 16-octet address in the exchange instructions.
S7 - Use of the compressed form of header of the instruction (OPR_LENGTH < > %b111) is allowed
S8 - Use of the extension form of header of the instruction (OPR_LENGTH = %b111) is allowed
S9 - Use of the extension headers with the data field up to 254 octets of length.
S10 - Use of the extension headers with the data field up to 4 * 10^9 octets of length.
S11-S15 Maximal length of the data field in operands in the 4 octet words. These bits are the common field. Maximal length in octets is computed under the formula:

<max length> = (<value of this field> + 1) * 4.
If the value is equal %b1111, maximal length of the data is defined by the instruction format.

S16-S19 These bits are the common field. In the profile required from the addressee of the instruction, this field contains the version of the UMSP. It must is set to the value %b0001. In the profile given sender of the instruction, this field contains priority of the job. The more is value of this field, the more priority. The priority of the job is used:
- o In queues on sending to the transport layer for the instructions of the job.
- o For set of sending priority of the transport layer.
- o For set of computing priority of the task.

S20 - making the border multiple of 4 octets. If S16 = 1:
(1) OPR_LENGTH = %b111
(2) Each extension header and the field of operands begin with the border multiple of four octets.
(3) The necessary number of zero octets is added in the end of each header.

S21 - Use of the procedures name of objects.
S22 - Use of the objects name.

The permissible instructions:

S23 - The response of the execution on VM (instruction RSP) is provided.
S24 - Use of data reading and comparison instructions.
S25 - Use of data writing instructions.
S26 - Use of control transfer instructions.
S27 - Use of synchronize instruction.
S28 - Use of instructions of work witch objects.
S29 - Use of the immediate access to memory of object. If this flag is set to 0, the access to object is solved only through its procedures. If S28=0, this flag must be set to 0.
S30 - Use of instruction MVRUN in zero-session.
S31 - Reserved. Must have set to 0.

5.4 Session Closing

Initiate closing session connection the node must only, which has initiated its establishment. It uses the SESSION_CLOSE instruction for this purpose. The procedure of break of connection is 3-way handshake. The procedure of unconditional emergency end of connection is stipulated. It can be transmitted by any node.

Let node A is the initiator of the establishment of a session, and the node B is the second side of connection. The node A must send the instruction SESSION_CLOSE for closing session. The node A may recommence sending of the instructions after sending of this instruction. It means that it has refused closing connection. The instructions of response (see section 6) does not influence on the closing of connection. The node, which has sent SESSION_CLOSE, does not use the timeout and can be waiting for the response beyond all bounds long.

The node B, after reception of the instruction SESSION_CLOSE, sends in the answer the instruction RSP_P. The zero basic return code responds closing session. The non-zero basic return code cancels closing session. After sending of positive response, the node must not use connection during 30-second timeout. If the instruction SESSION_ABEND or any other instruction, except response instruction, has not been received from the node A after the expiration of this time, the node send the instruction SESSION_ABEND and considers the session connection closed.

The node A sends the instruction SESSION_ABEND after reception of positive response on the instruction SESSION_CLOSE. After that, the connection is considered closed. The node A may refuse closing of connection. For this purpose, any instruction is sent, including NOP. In this case, the procedure of end interrupts, and the session connection is translated in the working state.

5.4.1 SESSION_CLOSE

The instruction "To close the session" (SESSION_CLOSE) initiates the end of session connection. It has the following values of fields:

```
OPCODE = 15
PCK = %b01/11
CHN = 0
ASK = 0
EXT = 0/1
OPR_LENGTH = 0/1
SESSION_ID - Contains the session identifier assigned by the
             addressee.
Operands:
   2 octets: The basic termination code.
   2 octets: The additional termination code.
The optional extension header:
   _MSG - contains the arbitrary message.
```

The operands may be absent. It is equivalent to the zero exit code.

5.4.2 SESSION_ABEND

The instruction "Abend of session" SESSION_ABEND is applied to unconditional end of session. The node, which has sent this instruction, finishes the exchange of the data on connection at both sides, not waiting responses from other node. The instruction has the following values of fields:

```
OPCODE = 16
PCK = %b01/11
CHN = 0
ASK = 0
EXT = 0/1
OPR_LENGTH = 0/1
SESSION_ID - Contains the session identifier assigned by the
             addressee.
Operands:
   2 octets: The basic termination code.
   2 octets: The additional termination code.
The optional extension header:
   _MSG - contains the arbitrary message.
```

The operands may be absent. It is equivalent to the zero termination codes.

5.5 Task Termination

The task is finished during the process of the job finishing at the normal end of the user application working. This procedure is described in the following item. The following situations require finishing the task irrespective of the job:

- There are not enough of computing resources for maintenance of the task on the node;
- The node finishes the work;
- If VM has accepted such decision for the internal reasons.

The references to the resources allocated by the task can be on any node, on which the job is carried out. Therefore, all nodes must be notified of the end of the task.

Node, finishing the task, must abnormally close all session connections joining the finished task (to send the instruction SESSION_ABEND).

5.5.1 TASK_TERMINATE

The instruction "To terminate the task" (TASK_TERMINATE) is sent from the node, on which the task is finished, to JCP. The instruction has the following values of fields:

```
OPCODE = 17
PCK = %b00
CHN = 0
ASK = 0
EXT = 0/1
OPR_LENGTH = 2/3  ; Depends on the length of CTID.
Operands:
   2 octets: The basic termination code.
   2 octets: The additional termination code.
   4/8 octets: CTID.
The optional extension header:
   _MSG - contains the arbitrary message.
```

After sending of the instruction TASK_TERMINATE to JCP, the node sends the instruction of unconditional end of connection ABEND_SESSION on all session connections connected with a task. After that, the task is considered completed.

If the basic return code in the instruction TASK_TERMINATE is equal to 0, it is not required to notify other nodes about the end of the task. Such situation arises, if the task did not allocate dynamic resources. If the basic return code is unequal to 0, JCP must notify about the task end the other nodes, on which the job is carried out, after reception of the instruction TASK_TERMINATE. JCP responds for the notification of all nodes of the job about the task end.

5.5.2 TASK_TERMINATE_INFO

The instruction "The information on terminating of the task" (TASK_TERMINATE_INFO) is used for the notification about the task end. It is sent from JCP to other nodes, on which the job is carried out. The instruction has the following values of fields:

```
OPCODE = 18
PCK = %b00
CHN = 0
ASK = 0
EXT = 0/1
OPR_LENGTH = 2-5 ; Depends on the length of GTID.
Operands:
   2 octets: The basic termination code.
   2 octets: The additional termination code.
```

4-16 octets: GTID of the terminated task. JCP forms GTID from LTID (from the instruction TASK_REG) and address of transport layer of the task.

The optional extension header:

_MSG - contains the arbitrary message.

The fields of termination codes are taken from the instruction TASK_TERMINATE. The job must delete (to make invalid) all references to resources concerning the node, on which the completed task worked, at reception of the instruction TASK_TERMINATE_INFO.

5.6 Job Completion

The job is finished, when the appropriated to it the user application on the node, on which it was initiated, is finished. The end of the job occurs under the initiative of VM. Besides, it can be completed under the JCP initiative at ending the lifetime of the job or at end of the JCP node working.

5.6.1 JOB_COMPLETED

The instruction "The task is completed" (JOB_COMPLETED) is sent from the node, which initiated the job, in the JCP side. It has the following values of fields:

```
OPCODE = 19
PCK = %b00
CHN = 0
ASK = 0
EXT = 0/1
OPR_LENGTH = 2/3 ; Depends on the CTID length.
Operands:
   2 octets: The basic completion code.
   2 octets: The additional completion code.
   4/8 octets: CTID of the completed task of the job. CTID is a
               part GJID of the job.
The optional extension header:
   _MSG - contains the arbitrary message.
```

After sending of the instruction JOB_COMPLETED to JCP, the node sends on all connected with the session connections of the job the instruction of unconditional end of connection ABEND_SESSION. After that, the job is considered completed.

JCP must notify of the end of the job the nodes, on which the job is carried out, after reception of the instruction JOB_COMPLETED. JCP responds for the notification of all nodes of the job about end of the job.

The instruction TASK_TERMINATE_INFO may be transferred under the initiative JCP, if node of the task has abnormal terminated work.

5.6.2 JOB_COMPLETED_INFO

The instruction "The information on completion of the job" (JOB_COMPLETED_INFO) is used for the notification about end of the job. It is sent from JCP to other nodes, on which the job is carried out. The instruction has the following values of fields:

```
OPCODE = 20
PCK = %b00
CHN = 0
ASK = 0
EXT = 0/1 ;
OPR_LENGTH = 2-5 ; Depends on the GJID length and presence of
                   fields completion code.
Operands:
   2 octets: The basic completion code.
   2 octets: The additional completion code.
   4-16 octets: GJID of the completed job.
The optional extension header:
   _MSG - contains the arbitrary message.
```

The fields of completion codes are optional.

The fields of completion codes are taken from the instruction JOB_COMPLETED. At reception of the instruction, JOB_COMPLETED_INFO the node must make the following:

(1) To remove all session connections, connected to the task. At that, it is not necessary to send network primitives.
(2) To abnormally finish the task of the job and to deallocate all dynamic resources of the task.

The instruction JOB_COMPLETED_INFO is used for the end of the job under the JCP initiative at the end of lifetime or at end of the JCP node working. In these cases, the node initiated the job is the first addressee of the instruction.

JCP considers the job completed after sending of all instructions JOB_COMPLETED_INFO.

5.7 Activity Control of Nodes

UMSP unites nodes, which have any arrangement in the network and which are not having uniform controls. Each of nodes can be disconnected or reloaded at any moment of time. However, other nodes

can be not notified about it. The fact of breaking or repeated establishment of transport connection cannot be the indicator of disconnect or restart of the node. The control of transport connections is not the part of the UMSP protocol and the presence of transport connection is not obligatory.

Besides the separate task on the node can be finished emergency. Procedure described in section 5.5.1 in this case must be executed. If this procedure cannot be executed, must is abnormally finished work of the node.

The JCP executes the functions of the control of nodes activity. The instruction of request of the status TASK_REQ is sent periodically between tasks on nodes and JCP for this purpose.

The following actions JCP are possible at detection of deactivating of the node:

(1) If the task initiated the job was finished, it is considered, that the job is completed. JCP sends the instruction JOB_COMPLETED_INFO to all other nodes, on which the job was executed.
(2) JCP sends the instruction TASK_TERMINATE_INFO to all other nodes of the job, if the task, which has not initiated the job, is finished.

The deactivating of the JCP node imposes the restriction on GJID appropriated by it after reloading. The following variants are probable:

(1) The disconnection of the JCP node passed normally. It transferred to all nodes, which it has controlled, instruction JOB_COMPLETED_INFO. In this case, it can appropriate anyone GJID after reloading.
(2) There is the emergency disconnect of the JCP node. It has not informed all nodes about the deactivating. In this case, it must guarantee after reloading, that new GJID will not concur witch the GJID, existing up to the reload, during two maximal intervals of inactivity time (which sets this JCP).

The reload of nodes, which are not being JCP, does not impose restrictions on LTID established on these nodes.

5.7.1 _INACTION_TIME

The extension header "The time of inaction" (_INACTION_TIME) allows setting the inaction time of the node (non JCP). It has the following values of fields:

```
HEAD_CODE = 2
HEAD_LENGTH = 1;
HOB = 1
DATA contains:
   2 octets: The inaction period.  The number of 0,5 second
   intervals, through which the activity of the node - sender of
   the instruction from the side JCP - will be checked.
```

The inaction period must be more than three intervals of the maximal time of delivery at the transport layer. The waiting period in queue to the transport layer does not influence on timeout.

The header _INACTION_TIME may be attached to the following instructions:

(1) To the instruction TASK_REG. In this case must be satisfied condition - on node there must not be other active tasks, which are controlled the JCP of addressee. The zero-value specifies that the activity checking is unused. The absence of the header specifies that the inaction period must be set on the JCP.
(2) To the instruction TASK_REJECT, if the time from the instruction TASK_REG does not fit for JCP.
(3) To the instruction TASK_CONFIRM, if instruction TASK_REG had no this header.

If JCP receives the instruction TASK_REG with the attached heading _INACTION_TIME, it must check up presence of active tasks with sender node (as it can mean, that the node was reloaded). If such tasks exist, for each of them it is necessary to execute procedure of end of the task described in section 5.6.2. The instruction TASK_CONFIRM must be sent only after that.

5.7.2 STATE_REQ

The instruction "State Request" (STATE_REQ) is sent from JCP to the definite task of other node. The instruction has the following values of fields:

```
OPCODE = 21
PCK = %b00
CHN = 0
ASK = 0
EXT = 0
OPR_LENGTH = 1/2 ; Depends on the LTID length.
Operand:
   4/8 octets: LTID, established on the node of the instruction
               addressee.
```

The instruction STATE_REQ will be sent in the defined task but it has concern with node. It is sent, if between the node and JCP was not sending of the instruction during inactive time. The task activated after sending of last instruction STATE_REQ does not influence the control of activity.

The instruction TASK_STATE is sent in reply to STATE_REQ. At expectation of the response, the timeout equal to one inaction period is used. After the expiration of the timeout the node is considered switched - off.

If the node not receives of any instructions from JCP during two intervals of inaction time, it is considered, that JCP has finished the work. The actions of the node in this case are described in section 5.6.2 at receiving the instruction JOB_COMPLETED_INFO. The check of this condition is optional for the node.

If at JCP there are no active tasks connected with the defined node, the control of activity of this node will not be carried out.

5.7.3 TASK_STATE

The instruction "Task State" (TASK_STATE) is sent from the definite task to JCP. It serves for the response of the instruction STATE_REQ. The instruction has the following values of fields:

```
OPCODE = 22
PCK = %b00
CHN = 0
ASK = 0
EXT = 0
OPR_LENGTH = 1/2/3 ; Depends on the CTID length.
Operands:
   1 octet:  The state code of task.  The following values are
             defined for this field:
      %x01 - The task is active and has active session
             connections.
      %x02 - The task is active and have no session connections.
      %x03 - The task is active, have no session connections and
             have no resources, allocated on the node.
      %x04 - The task is completed.
   1/3 octets: Reserved.  If OPR_LENGTH = 1, then this field has
               length 1 octet, else 3 octets. JCP must not check
               the value of this field.  It is established in zero
               value by sending.
   2/4/8 octets: CTID connected with LTID from the instruction
                 STATE_REQ.
```

If OPR_LENGTH = 1 that length of the reserved field is equal to one octet and length CTID makes two octets. In all other cases, length of the reserved field is equal 3 octets and length CTID - not less than 4 octets.

5.7.4 NODE_RELOAD

The instruction "The node was reloaded" (NODE_RELOAD) is sent to JCP as the negative response to STATE_REQ instruction. NODE_RELOAD has the following values of fields:

```
OPCODE = 23
PCK = %b00
CHN = 0
ASK = 0
EXT = 0
OPR_LENGTH = 1/2 ; Depends on the LTID length.
Operands:
   4/8 octets: LTID.  The value is taken from the instruction
               STATE_REQ.
```

The instruction RELOAD_NODE indicates, that the task with given LTID for given JCP on the node is absent. At reception of this instruction, JCP must make the following:

(1) To send the instruction STATE_REQ to all tasks of the node, which were initiated before a sending of the penultimate instruction STATE_REQ.
(2) To wait for ending of one inaction interval after sending of the last instruction STATE_REQ (on which the negative response is received).
(3) To send the instructions STATE_REQ to all tasks of the node, which were initiated between last and penultimate instructions STATE_REQ (not including instructions from item 1).

For all instructions STATE_REQ the positive response (TASK_STATE) or negative response (RELOAD_NODE) must be transmitted.

5.8 Work without session connection

The protocol provides the data exchange between nodes without an establishment of session connection. In this case, initialization of the job and tasks is not made and JCP is not used.

The format of the instructions, transmitted without the establishment of connection, is completely correspond to the instructions transmitted by session connections. The difference is that the field SESSION_ID has zero value or PCK = %b00.

The node, supporting work without the establishment of session connection, must have VM, which executes by default the instructions transmitted without the establishment of connection. In fact, these instructions are executed within the framework of a so-called zero-session (or zero-task) of this VM. The memory address space of this VM is accessible without a connection establishment.

The instruction SESSION_INIT with SESSION_ID = 0 and REQ_ID = 0 allows to specify parameters of its zero-session and to request the zero-session parameters of the addressee node. If the node, which has received such instruction, provides the requiring profile, it sends the instruction SESSION_ACCEPT. If the profile is not provided, the answerback instruction SESSION_INIT will send, in which the field SESSION_ID and REQ_ID also have the value 0. Actually, such instructions of session initialization do not establish connection, but have the information meaning. The exchange of the data by zero-session can occur irrespective of its.

There are the following restrictions at working without connection:

- The chain must be sent, only if it is completely located in one segment of the transport layer.
- It is impossible to request an allocation of memory and to create objects (except instruction MVRUN). This objects is not adhered to the definite job and is not automatically release the resources at the end of the job, which has created them.
- Parameters of functions and the returned values must not contain the pointers, because the node can be reloaded at any moment. It will result that the pointers will become invalid or will address other objects.

The protocol cannot check those conditions. Their realization lays on VM wholly.

The work without establishment of session connection may be used in the following systems:

- In simple devices, which do not have the operational system;
- On servers which are executed a plenty of requests (for work without connection of resources is used less);
- In systems requiring the fast response to rare requests (if keeping of connection is inexpedient).

6 Instructions of Exchange between VM

The instructions intended for an exchange between VM uses values OPCODE in range 128 - 223. Depending on length of the operands field, several formats of the instruction may be defined for one OPCODE. The complete instruction format is defined by aggregate of the values of fields OPCODE and OPR_LENGTH.

The instruction has the field REQ_ID, if in the instruction header flag ASK = 1. REQ_ID is used for the response identification. The value of this field is specifies by VM. The response is formed by VM, too. The protocol does not check the response and does not analyze the value of the field REQ_ID for the instructions of exchange between VM. One of the instructions RSP, DATA, RETURN, ADDRESS, OBJECT or PROC_NUM is used for sending of the response. The instructions of response have ASK = 1 and the value taken from the confirmed instruction is record in REQ_ID. The instructions of response do not require the response.

The instructions of exchange between VM may be sent through UDP at observance of the following conditions:

- ASK = 0;
- The instruction is located in one segment UDP;

The timeouts and the repeated sending are not used at UMSP layer for instructions of exchange between VM. It is explained to, that the time of sending instructions with low priority may be very large because of the output queues. Therefore, the VM must make a decision on timeout, as only VM has the complete information on type of the transmitted data. Besides, the transport layer protocol must use the timeouts.

A few VM may be connected to the protocol on the node. VM may simultaneously execute several jobs. Each job may work in its address space. The protocol determines VM and job, which the received instruction must transfer to, on field SESSION_ID value.

The local memory address is located in the instruction in field of length 2/4/8 octets. If memory address length in the instruction is not equal to memory address length defined for the node, the following variants are possible:

- If memory address length is set in 24 bits for the node, the address is writes in the end of 4 - octets field. The 0 value sets in an initial (zero) octet.

- o If the instruction format assumes the memory address length not less than 4 octets, 2-octet address is located in the last octets. The first 2 octets must set to zero.
- o If instruction is the member of a chain and it has the less length of the memory address, than it is defined for the node - it is considered, that the base-displacement addressing is used. If the value of the memory base is not assigned for the chain - instruction is erroneous.
- o If the instruction is not the member of a chain and has the length of memory address less, than it is defined for the node, it is considered, that the abbreviated address is used. The complete address length must be received by padding in front of it the necessary number of zero-value octets.
- o In all other cases, the instruction is erroneous.

Complete 128-bit memory address writes in operands in the 16-octets field. The reason of using of the complete address is that the additional information, using by the memory control subsystem in the node, may contain in its field FREE (see section 2.1). If the FREE of the complete address is set to zero, it is recommended to use local address in operands.

Operands field has a length, which is an integral number of 32 bits. The alignment is making by padding, if necessary, of the zero-value octets at the end of the field.

Header fields of the instructions not defined in the formats description are used according to the description from section 3.

The instruction of the transfer control JUMP, CALL, CALL_BNUM and CALL_BNAME may contain the information about VM of the sender. If VM type and VM version of the sender are contains in the instruction, the call parameters are formed in a format VM of the sender. Else, the call parameters have format defined by VM of the addressee. The code is always connected with of specific VM.

All instructions of the protocol work with binary data and do not provide operations of formats transformation.

6.1 Data Reading/Writing Instructions

6.1.1 REQ_DATA

The instruction "To request a data" (REQ_DATA) is used for the data request from the remote node. Two instructions REQ_DATA with length of the length field 2 and 4 octets are defined. These instructions have the following values of fields:

```
      OPCODE = 130/131      ; For length of the length field of 2/4
                              octets.
      OPR_LENGTH = 1/2/3/5  ; Depends on address length.
      Operands:
         2/4 octets: The length field. The number of the required data in
                     octets.
         2/4/8/16 octets: The memory address of the required data.
```

The instruction DATA, containing required data, is sent in reply to it. If the data cannot be sent, the instruction RSP with the non-zero basic return code, comes back.

6.1.2 DATA

The instruction "The data" (DATA) is sent in reply to the instruction REQ_DATA and OBJ_REQ_DATA. The instruction has the following values of fields:

```
      OPCODE = 132
      OPR_LENGTH = 0 - 65535  ; Depends on the immediate data length of
                                the operand.
      Operands:
         0 - 262140 octets: Immediate data.  If OPR_LENGTH = 0, this
                            field are absent.
      Extension headers:
         _DATA - Contains immediate data.  If OPR_LENGTH <> 0, this
                 header are absent.
```

The extension header is used, if the data are more then an maximum operands field size. The data must not be sent simultaneously in operands and in the extension header. To make the length of data multiple of 4 octets, 1 - 3 zero-value octets are padded in the end of a field.

6.1.3 WRITE

The instruction "To write the data" (WRITE) is used for data writing on the remote node. The instruction has the following values of fields:

```
      OPCODE = 133/134/135/136  ; For memory address length of 2/4/8/16
                                  octets.
      OPR_LENGTH = 1 - 65535    ; Depends on length of the immediate
                                  data.
      Operands:
         2/4/8/16 octets: The memory address for writing the data.
         0 - 262136 octets: Immediate data for write.
```

```
      Extension headers:
         _DATA - Contains immediate data.  This header is present only,
                 if the data does not contain in operands.
```

At address length of 2 octets the data length must be 2 octets. In all other cases, address length must be not less than 4 octets and data length must be multiple of 4 octets. The data must not be sent simultaneously in operands and in the extension header.

The instruction RSP is sent in reply to the instruction WRITE. The zero basic return code defines normal executing.

6.1.4 WRITE_EXT

The instruction "The extension writing of data" (WRITE_EXT) is used for the data writing on the remote node. Length of the data may be 1 - 262132 octets with a step 1 octet. The instruction has the following values of fields:

```
      OPCODE = 137
      OPR_LENGTH = 3 - 65535  ; Depends on length of the immediate data.
      Operands:
         1 octets: Always set to zero.
         3 octets: The number of the write data in octets.  The zero-
                   value is not available.
         4 - 262132 octets: Immediate data for write.  The data length
                            must be multiple of 4 octets.
         4/8/16 octets: The memory address for writing the data.
```

To make the immediate data multiple of four octets, the data is padded with 1 - 3 zero-value octets at the end of a field.

The instruction RSP is sent in reply to the instruction WRITE_EXT. The zero basic return code defines normal executing.

6.2 Comparison Instructions

6.2.1 CMP

The instruction "To compare" (CMP) is used for binary data comparison. It has the following values of fields:

```
      OPCODE = 138/139/140/141  ; For the address length of 2/4/8/16
                                  octets.
      OPR_LENGTH = 1 - 65535    ; Depends on length of the immediate
                                  data.
```

```
      Operands:
         2/4/8/16 octets: The memory address for compared data.
         2 - 262136 octets: The immediate data for the comparison.
```

At the address length of 2 octets the data length must be 2 octets. In all other cases length of the address must not be less than 4 octets and the data length is multiple to four octets.

6.2.2 CMP_EXT

The instruction "The extension compare" (CMP_EXT) is used for binary data comparison. Length of the data may be 1 - 262132 octets with a step 1 octet. The instruction has the following values of fields:

```
      OPCODE = 142
      OPR_LENGTH = 3 - 65535  ; Depends on length of the immediate data
                                and the address.
      Operands:
         1 octet: Always set to 0.
         3 octets: The length of compared data in octets.  The zero-value
                   is not available.
         4 - 262132 octets: The immediate data for the comparison.  The
                            length of field is multiple of 4 octets.
         4/8/16 octets: The memory address of compared data.
```

To make the immediate data multiple of four octets, the data is padded with 1 - 3 zero-value octets at the end of a field.

6.2.3 Response to Comparison Instructions

The instruction RSP is sent in reply to the instruction CMP, CMP_EXT and OBJ_CMP (see below). If the comparison was executed, the basic return code is equal to zero. The additional return code is equal to -1, if the data at the address memories are less then the data from the operand; 0, if they are equal; and 1, if they are more. If the comparison cannot be executed, the basic return code of the instruction RSP must be non-zero.

6.3 Control Transfer Instructions

6.3.1 JUMP, CALL

The "Unconditional jump" (JUMP) and "To Call-subroutine" (CALL)_ instructions have an equal format and differ only by OPCODE. These instructions have the following values of fields:

```
      OPCODE = 143/144  ; Correspondingly for the JUMP not containing
                          and containing the information about VM.
```

```
            145/146  ; Correspondingly the CALL not containing and
                       containing the information about VM.
      OPR_LENGTH = 2 - 65535  ; Depends on inclusion of the information
                                about VM, address length and parameters
                                length.
      Operands:
         2 octets: The VM type of the sender.  If OPCODE=143/145 this
                   field is absent.
         2 octets: The VM version of the sender.  If OPCODE=143/145 this
                   field is absent.
         4/8/16 octets: The address of memory, where is necessary to
                        transfer control.
         2 octets: The number of 32 bit words in the call parameters
                   field.
         4 - 262134 octets: The immediate data are the parameters of a
                            call.
```

On the reception side the processing of the instructions of a control transfer occurs as follows:

- The memory address is checked. If it has erroneous value, the negative response RSP is sent. At this stage, the correctness of parameters of a call may be also checked up.
- If the memory address and the parameters of a call have correct value, the positive response RSP is sent for the instruction JUMP. The transmitting side considers the instruction JUMP executed after receiving response.
- For response of an execution of the instruction CALL the instruction RETURN is sent. The instruction RETURN may contain the returned values. If there is an exception condition in a thread of control created by the CALL instruction, the instruction RSP with a non-zero basic return code is sent instead of RETURN.

6.3.2 RETURN

The instruction "Return of control" (RETURN) is used at return of control from the instructions CALL, MVRUN, CALL_BNUM and CALL_BNAME (see below). Those instructions have the following values of fields:

```
   OPCODE = 147
   OPR_LENGTH = 0 - 65535  ; Depends on length of the immediate data.
   Operands:
      0 - 262140 octets: Immediate data returned from the subroutine.
```

If it is not required to send returned value, the instruction RETURN does not contain operands. The data format coincides with the instruction, for which the response (format VM of the sender or addressee) will be sent.

6.4 Memory Control Instructions

UMSP gives means for division of memory for a code and for the data. The protocol does not make checks of correctness of operations with memory. The code and the data use common address space. The control of memory is completely realized by VM.

6.4.1 MEM_ALLOC

The instruction "To allocate a memory for the data" (MEM_ALLOC) is used for request of the allocation of memory under the data. The instruction has the following values of fields:

```
OPCODE = 148
OPR_LENGTH = 1
Operands:
   4 octets: The size of required memory in bytes.
```

For the positive response on the instruction MEM_ALLOC, the instruction ADDRESS, for negative - RSP with the non-zero basic return code is sent. The received address can be used by the protocol in the instructions of reading/writing, comparison and synchronization.

6.4.2 MVCODE

The instruction "To move the code" (MVCODE) is used for moving of the executable code from one node on another. The instruction has the following values of fields:

```
OPCODE = 149
OPR_LENGTH = 1 - 65535 ; Depends on length of the code field.
Operands:
   2 octets: The VM type of addressee.
   2 octets: The VM version of addressee.
   0-262136 octets: contains the executable code.
The extension headers:
   _DATA - contains the executable code.  This header is present
           only, if the code does not contain in operands.
```

The code is always connected with VM of the definite type. The code field is always transparent for the protocol. It is formed by the VM of sender and must contain all the information necessary VM of the receiver. The code must not simultaneously be sent in operands and in the extension header.

For the positive response on the instruction MVCODE, the instruction ADDRESS, for negative - RSP with the non-zero basic return code is used. The code transferred on the instruction MVCODE, may be executed by the instruction JUMP or CALL.

6.4.3 ADDRESS

The instruction "The memory address" (ADDRESS) is used for the positive response on the instruction MEM_ALLOC and MVCODE. ADDRESS has the following values of fields:

```
OPCODE = 150
OPR_LENGTH = 1/2/4; Depends on length of the address.
Operands:
   4/8/16 octets: The address of the allocated memory.
```

For the instruction, MEM_ALLOC the address specifies first byte of the allocated data area. For the instruction MVCODE the contents of the address is defined VM, by which the code is connected.

6.4.4 FREE

The memory allocated with the instructions MEM_ALLOC and MVCODE, must be explicitly release. For this purpose, the instruction "To free the memory" (FREE) is used. It has the following values of fields:

```
OPCODE = 151
OPR_LENGTH = 1/2/4; Depends on length of the address
Operands:
   4/8/16 octets: the address of free memory.
```

VM must free this memory automatically at end of the task on the node.

6.4.5 MVRUN

The instruction "To move and run" (MVRUN) is used for simultaneous move of a code and its execution. The instruction has the following values of fields:

```
OPCODE = 152
OPR_LENGTH = 1 - 65535 ; Depends on length of the code field.
Operands:
   2 octets: The addressee VM type.
   2 octets: The addressee VM version.
   4 - 262136 octets: Contains an executable code.
```

The extension headers:
_DATA - Contains an executable code. This header is present only, if the code does not contain in operands.

The executable code is the transparent buffer with the binary data for the protocol. The format of this field is defined by the VM and it must contain all the information necessary for the loader VM of the addressee, including parameters of a call.

The code must not simultaneously be sent in operands and in the extension header.

The answer to the instruction MVRUN is formed similarly to instruction CALL. It is not necessary to release memory allocated for a code by this instruction. The memory must deallocate the VM.

6.5 Other Instructions

6.5.1 SYN

The instruction "To Synchronize" (SYN) is used for the single message about the data change. The instruction has the following values of fields:

OPCODE = 153/154/155 ; For length of the address 4/8/16 octets.
OPR_LENGTH = 2 - 65535; Depends on length of the data
Operands:
4/8/16 octets: The memory address of the tracking data.
2 - 131068 octets: The initial data. Length of the data must be multiple of two octets.
2 - 131068 octets: A mask for comparison. Length of this field is equal to length of a field of the initial data.

The tracking data is set by the memory address in the first operand. These data are originally compared to the initial data value from the second operand. If the values do not coincide, it is considered, that the data have changed. The third operand allows setting a mask for comparison. Set to one bits of the mask specifies bits in the data, which change must be traced.

The following variants of the answer are probable on the instruction:

- If the address of local memory is incorrect, the instruction RSP with the non-zero basic return code is sent for the response.
- If the data do not change, in the response nothing is sent.
- If the data have changed, the instruction DATA with new value of the traced data is sent.

6.5.2 NOP

The instruction "No operation" (NOP) has the following values of fields:

```
OPCODE = 156
OPR_LENGTH = 0 - 65535
Operands:
   0 - 262140 octets: Encapsulated data.
Extension headers:
   Any Extension headers.
```

The instruction NOP is intended for the decision of the following tasks:

- Send the control extension headers, when there are no other instructions for sending in a session
- Encapsulate the fragmented instructions and transactions with the established flag of special processing (see section 7).

6.6 Work with Objects

The protocol has a set of the instructions being expansion of the protocol RPC [6]. As against RPC, UMSP allows immediately to address memory on remote nodes and to send the pointers in parameters and returned values.

The UMSP object is identified by the 4-octet number. The values are divided into the following ranges:

```
 I -> %x00000000 - 1FFFFFFF   are assigned for standard objects
II -> %x20000000 - 3FFFFFFF   are assigned for users objects
III -> %x30000000 - 4FFFFFFF  free
 IV -> %x50000000 - DFFFFFFF  transient
  V -> %xE0000000 - FFFFFFFF  reserved
```

The objects from a range I must be definite, as standard, and the specifications of their interfaces must be published. The protocol does not suppose the private or not described interfaces of standard objects.

The objects from a range II must be registered, but the specifications of their interfaces may be optional published. These numbers are applied in cases, when it is required to exclude the probable conflict of systems of the different manufacturers.

The range III can be used freely. The objects accessible on these numbers may be created statically or dynamically. These objects can have any interfaces.

All objects, concerning ranges I, II and III, is common for all jobs on the node, including zero-session. Their interfaces are accessible to all tasks on the node, depending on parameters of authentication.

The range IV is intended for objects created dynamically within the framework of one job. These objects are the isolated associative memory of the job. The access to these objects must be granted only for one job. The zero-session has no access to these objects.

The protocol grants the access to the data of object, as to the continuous segment of memory. The memory of objects may be overlapping or no overlapping with flat local memory of the node. The offset field is used in the instructions of work with the data of object. The offset rules writing are similar to the local address rules writing.

The address memory length of the node, definite for the UMSP protocol, limits the maximal data size of one object. The instructions definite in the given section, allow to work with associative memory with the theoretical limiting size on one node - 2^96 (7,9 * 10^28) Byte.

In addition to the number, the object has the version, 2 octets length, and realization, 2 octets length. The protocol requires obligatory compatibility from bottom-up for all realizations of one version of object. The publication of new realization of standard object may contain only added interfaces.

If for the sender of the instruction the version and/or the realization of object do not play any role or is unknown, the instruction may contain zero fields of the version and realization of object or only zero field of realization. The zero field of the version and non-zero field of realization are not allowed.

6.6.1 Reading/Writing of the Objects Data

6.6.1.1 OBJ_REQ_DATA

The instruction "To request the data of object" (OBJ_REQ_DATA) is used for request of data of the Object from the remote node. The instruction has the following values of fields:

```
OPCODE = 192/193   ; For length of the field of length 2/4 octets.
OPR_LENGTH = 3/4/5 ; Depends on length of the offset field.
Operands:
   4 octets: The number of object.
   2 octets: The version of object.
   2 octets: The realization of object.
   2/4 octets: The length of the required data in octets.
   2/4/8 octets: Offset required data from the beginning of object
                 in bytes.
```

At length of the length field of 2 octets the offset length must be 2 octets. In all other cases, length of the length field and offset length must be not less than 4 octets.

The instruction DATA, containing the required data, is sent for reply to instruction OBJ_REQ_DATA. If the data cannot be transmitted, the instruction RSP from the non-zero basic return code comes back.

6.6.1.2 OBJ_WRITE

The instruction "To write the data in object" (OBJ_WRITE) is used for write of the data in object. The instruction has the following values of fields:

```
OPCODE = 194/195/196   ; For length of the offset field of 2/4/8
                         octets.
OPR_LENGTH = 3 - 65535 ; Depends on the data length.
Operands:
   4 octets: The number of object.
   2 octets: The version of object.
   2 octets: The realization of object.
   2/4/8 octets: The offset in object for the data writes.
   2 - 262128 octets: The immediate data for write.
The extension headers:
   _DATA - Contains immediate data for write. This header is
           present, only if the data is not present in operands.
```

At length of the field-offset of 2 octets, length of the data must be 2 octets. In all other cases, the offset length must be not less than 4 octets and the data length is multiple to four. The data must not simultaneously be sent in operands and in the extension header.

The instruction RSP is sent in reply to the instructions OBJ_WRITE. The zero basic return code defines the normal execution.

6.6.1.3 OBJ_WRITE_EXT

The instruction "The extension writing of the data in object" (OBJ_WRITE_EXT) is used for write of the data in object. Length of the data may be 1 - 262132 octets with the step 1 octet. The instruction has the following values of fields:

```
OPCODE = 197
OPR_LENGTH = 3 - 65535; Depends on the data length and the address
                        length.
Operands:
   4 octets: The number of object.
   2 octets: The version of object.
   2 octets: The realization of object.
   1 octet: Always set to 0.
   3 octets: Length written down data in octets. The zero-value is
             incorrect.
   4 - 262124 octets: The immediate data for write. Length of the
                      data is multiple of 4 octets.
   2/4/8 octets: Offset in object for the data write.
```

If the length of the written down data is not multiple of four octets, the data is padded with 1 - 3 zero octets at the end.

The instruction RSP is sent in reply to the instructions OBJ_WRITE_EXT. The zero basic return code defines the normal execution.

6.6.2 Comparison Instructions of the Objects Data

6.6.2.1 OBJ_DATA_CMP

The instruction "To compare the data of object" (OBJ_DATA_CMP) is used for binary comparison of data of the object by the immediate data from operands. The instruction has the following values of fields:

```
OPCODE = 198/199/200  ; For length of offset field of 2/4/8
                        octets.
OPR_LENGTH = 3 - 65535; Depends on length of the data.
Operands:
   4 octets: The number of object.
   2 octets: The version of object.
   2 octets: The realization of object.
   2/4/8 octets: Offset in object for the compared data.
   2 - 262128 octets: The immediate data for comparison.
```

At length of a field of 2 octets offset the data length must be 2 octets. In all other cases the offset length must be not less than 4 octets and the data length is multiple to 4 octets.

The response to the instruction OBJ_DATA_CMP is described in section 6.2.3.

6.6.2.2 OBJ_DATA_CMP_EXT

The instruction "The extension compare of data of the object" (OBJ_DATA_CMP_EXT) is used for binary comparison of data of the object by the immediate data from operands. Length of the data may be 1 - 262132 octets with a step 1 octet. The instruction has following values of fields:

```
OPCODE = 201
OPR_LENGTH = 5 - 65535  ; Depends on length of the immediate data
                          and the address length.
Operands:
   4 octets: The number of object.
   2 octets: The version of object.
   2 octets: The realization of object.
   1 octet: Always set to 0.
   3 octets: The length of compared data in octets.  The zero-value
             is incorrect.
   4 - 262124 octets: The immediate data for the comparison.  The
                      length of field is multiple of 4 octets.
   4/8 octets: Offset in object for the compared data.
```

To make the immediate data multiple of four octets, the data is padded with 1 - 3 zero-value octets at the end.

The response to the instruction OBJ_DATA_CMP_EXT is described in section 6.2.3.

6.6.3 Execution of the Objects Procedures

6.6.3.1 CALL_BNUM

The instruction "To call the object procedure over number" (CALL_BNUM) transfers control to the object procedure over indication of the number. The instruction has following values of fields:

```
OPCODE = 202/203 ; Accordingly for the instructions not containing
                   and containing the information about VM.
OPR_LENGTH = 4 - 65535 ; Depends on inclusion of the information
                         about VM and call parameters length.
```

```
      Operands:
         2 octets: The VM type of the sender.  If OPCODE=202 this field
                   is absent.
         2 octets: The VM version of the sender.  If OPCODE=202 this
                   field is absent.
         4 octets: The number of object.
         2 octets: The version of object.
         2 octets: The realization of object.
         4 octets: The number of the called procedure.
         4 - 262128 octets: Parameters of the call.
```

The processing on the reception side is made similarly instructions CALL (see section 6.3.1).

6.6.3.2 CALL_BNAME

The instruction "To call the object procedure over name" (CALL_BNAME) transfers control to the object procedure over indication of the name. The instruction has following values of fields:

```
      OPCODE = 204/205  ; Accordingly for the instructions not
                          containing and containing the information
                          about VM.
      OPR_LENGTH = 3 - 65535 ; Depends on inclusion of the information
                               about VM and call parameters length.
      Operands:
         2 octets: The VM type of the sender.  If OPCODE=204 this field
                   is absent.
         2 octets: The VM version of the sender.  If OPCODE=204 this
                   field is absent.
         4 octets: The number of object.
         2 octets: The version of object.
         2 octets: The realization of object.
         4 - 262128 octets: Parameters of the call.
      The extension header:
         _NAME - Contains the name of the called procedure.
```

The processing on the reception side is made similarly instructions CALL (see section 6.3.1).

The names may have the procedures of the objects belonging to ranges III and IV. The procedures of the objects belonging to ranges I and II must not have a name on the UMSP layer. They must have the number only.

6.6.3.3 GET_NUM_PROC

The instruction "To get the name of object procedure" (GET_NUM_PROC) allows receiving number of the procedure for objects in ranges III and IV over procedure name. The instruction has following values of fields:

```
OPCODE = 206
OPR_LENGTH = 2
Operands:
   4 octets: The number of object.
   2 octets: The version of object.
   2 octets: The realization of object.
The extension header:
   _NAME - Contains procedure name.
```

For the positive response on the instruction GET_NUM_PROC, the instruction PROC_NUM, for negative - RSP with the non-zero basic return code is sent.

6.6.3.4 PROC_NUM

The instruction "The procedure number" (PROC_NUM) is sent in reply to the instruction GET_NUM_PROC. The instruction PROC_NUM has following values of fields:

```
OPCODE = 207
OPR_LENGTH = 3
Operands:
   4 octets: The number of object.
   2 octets: The version of object.
   2 octets: The realization of object.
   4 octets: The number of procedure.
```

6.6.4 The Objects Creation

The objects from the ranges I and II (standard and assigned for the user) cannot be created on the remote node through the UMSP interface. These objects must be created only through API of the VM. The objects from the ranges III and IV can be created on the remote node by the protocol instructions.

The realization of objects from the ranges I - III (not connected with the certain job) is difficult enough. The reason is that the different jobs can have the different address spaces of memory. The pointers must be processed in the context of the job, from which they are received. Besides, these objects must trace the end of the jobs

for deallocation of dynamic resources. The specified requirements impose essential restrictions on these objects. The protocol does not impose any restrictions on objects from the range IV.

Unique key identifying object on node, is number of object. To objects from the ranges, III and IV the name may be assigned. The objects from range I and II must not have names on the UMSP layer. Within the framework of one task must not be two objects having one number or one name.

6.6.4.1 NEW, SYS_NEW

The format of both instructions "New object" (NEW) and "New system object" (NEW_SYS) is similar. First instruction creates object in the range IV, second - in the range III. These instructions have the following values of fields:

```
OPCODE = 208/209; Accordingly for NEW/NEW_SYS.
OPR_LENGTH = 3
Operands:
   2 octets: The addressee VM type.
   2 octets: The addressee VM version.
   2 octets: The version of object.
   2 octets: The realization of object.
   4 - 262136 octets: Immediate data necessary for creation of
                      object.
The extension headers:
   _DATA - Contains immediate data, necessary for creation of
           object. This header is present, only if the data is not
           present in operands.
   _NAME - Contains the name of object. This header is optional.
```

The instruction NEW_SYS is used for the creation of object accessible from any job, NEW - for creation of object accessible only from its job. If the object is created, the instruction OBJECT is sent for the response. If the object cannot be created, the instruction RSP with the non-zero basic return code is sent.

The immediate data field is transparent for the protocol. It is formed by the sender VM and it must contain the information, which is necessary to the addressee VM for the creation of object. Data must not simultaneously be sent in operands and in the extension header.

The field SESSION_ID of the instruction cannot have the zero value. The dynamic object must be created only in the context of the definite job. The object is always created on VM, with which the session is connected.

The zero values of the version and the realizations of object means, that the object have no these values.

It is possible to register the name of object simultaneously with its creation. The name contains in the _NAME extension header.

All objects created upon the instructions NEW and NEW_SYS must be obviously deleted. VM must automatically delete all dynamic objects, created and not deleted by the task, at the end of the task.

6.6.4.2 OBJECT

The instruction "The Object" (OBJECT) is used for the positive response on the instruction NEW and NEW_SYS. The instruction OBJECT has following values of fields:

```
OPCODE = 210
OPR_LENGTH = 2
Operands:
   4 octets: The number of object.
   2 octets: The version of object.
   2 octets: The realization of object.
```

6.6.4.3 DELETE

The instruction "To delete the object" (DELETE) is used for the deleting of object created on the instruction NEW or NEW_SYS. The instruction DELETE has the following values of fields:

```
OPCODE = 211
OPR_LENGTH = 1
Operands:
   4 octets: number of object
```

The object may be deleted only from the job, which has created it. The instruction RSP is sent in reply to this instruction.

6.6.5 The Objects Identification

At registration of object on the node, it may be identify by the name, the length of 4 - 254 octets. The name contains the symbols ASCII. The following versions of the protocol may define other types of the name.

The name identifies with the number of object and is its synonym. The names of all active objects in one task on the node must be unique. Thus, all active objects from the range of number I - III

must have the unique names for all tasks on the node. The protocol allows receiving the number of object by the name and the name of object by the number.

6.6.5.1 OBJ_SEEK

The instruction "To seek the object" (OBJ_SEEK) is used for seek of number of the object by the name. It has the following values of fields:

```
OPCODE = 212
OPR_LENGTH = 0
The extension header:
   _NAME - contains the name of object for search.
```

If the object is found - the instruction OBJECT is sent in the answer. If the object is not found - the instruction RSP with the non-zero basic return code is sent for the response.

The instruction OBJ_SEEK may be sent broadcast through UDP. In this case, it concerns to zero-session. The instruction may contain the field REQ_ID for identification of answers. The positive responses in this case must be sent only. The response may be transmitted through UDP.

6.6.5.2 OBJ_GET_NAME

The instruction "To get a name of the object" (OBJ_GET_NAME) is used for get of the name of object by number. It has the following values of fields:

```
OPCODE = 213
OPR_LENGTH = 1
Operands:
   4 octets: number of object for getting
```

If the object is present - the instruction OBJECT with the extension header _NAME is sent for the response. If the object is not present - the instruction RSP with the non-zero basic return code is sent for the response.

7 Chains

The instructions, which will be sent on one session connection, can be unified in a chain. The chain is a group of the instructions relational with each other. In one session, several chains simultaneously can be transferred. The chains can be the following types:

- o The sequence.
- o The transaction
- o The fragmented instruction.

If the instruction is included into a chain, the flag CHN should be equal 1. The field CHAIN_NUMBER of header contains number of a chain, INSTR_NUMBER - serial instruction number in a chain, since 0. The numbering of chains is conducted by the protocol. In one session simultaneously can be transferred up to 65533 chains. Values of numbers of chains %x0000 and %xFFFF reserved by the protocol. One chain can contain up to 65535 instructions.

The instruction with a zero serial number INSTR_NUMBER should contain the extension header describing a chain. Each type of a chain has own initiating extension header.

_END_CHAIN. The extension header "End of the chain" is transferred in last instruction of chain, irrespective of type of the chain. It has the following values of fields:

```
HEAD_CODE = 6
HEAD_LENGTH = 0
HOB = 1
```

Number of a finished chain contains in a field CHAIN_NUMBER of the instruction header, to which the extension header is attached.

The instructions, included in chains, can be transferred through UDP only if all chain is located in one segment.

7.1 Sequence

The sequence is a type of a chain, which unites the instructions dependent from each other. The following instruction of a sequence can be executed on VM, only if have been executed previous. If the current instruction cannot be executed, all other instructions of the given sequence (already sent or expecting sending) simply cancel. Due to this, it is possible for one computing control thread not to wait for the current instruction positive end and to transfer following at once.

_BEGIN_SQ. The extension header "To begin a sequence" is transferred in the first instruction of the sequence. It has the following values of fields:

```
HEAD_CODE = 3
HEAD_LENGTH = 0
HOB = 1
```

Number of created chain is established in field CHAIN_NUMBER of the instruction header, to which the extension header is attached. The field INSTR_NUMBER must have value 0.

The initiator of creation of a sequence is VM. It is not obligatory that the sequence should have known length beforehand. It can be completed in any moment. If it is necessary to finish a sequence and there are no instructions for sending, the instruction NOP can be generated.

7.2 Transaction

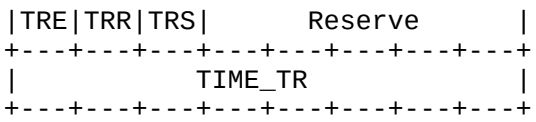
The transaction is a type of the chain uniting some possibly not connected with each other instructions. All transaction instructions must be executed all at once or must not be executed. It is possible to cancel or to confirm transaction execute. The transaction cancellation after execution is not stipulated. If it is necessary, such mechanism should be realized at VM level, because there can be instructions in transaction, which are impossible to cancel, for example a control transfer.

The initiator of transaction creation is VM. The transaction length must be known beforehand. The length will define a way of transaction transfer. It is connected with buffering described in section 7.4.

7.2.1 _BEGIN_TR

The extension header "To begin a transaction" _BEGIN_TR is transferred in the first transaction instruction. It has the following values of fields:

HEAD_CODE = 4
HEAD_LENGTH = 1
HOB = 1
DATA - Has the following format:



TRE

1 bit. The flag of obligatory execution. This flag relates only to completely transferred, but have not yet executed transaction. If TRE = 1, the transaction must be executed at

the expiration of existence time, established by field TIME_TR, or at emergency session end. If TRE = 0, at end of existence time the transaction must be cancelled and the negative acknowledgement must be transferred, and at emergency session end - must be simply cancelled.

TRR

1 bit. The flag of execution after sending. If TRR = 1, the transaction must be executed after sending of all instructions, of which it is consists, at once. Such transaction is executed after reception of the instruction with the extension header _END_CHAIN. If TRR = 0, it is necessary to transfer the special instruction EXEC_TR of transaction acknowledgement for its execution.

TRT

1 bit. The flag of special processing. It is entered for a possibility of the further expansion of the protocol. If TRT = 1, before transaction execution it is necessary to make some additional actions above the instructions, of which it is consists, for example to decipher. These actions can be definite in the additional extension headers transmitted in the transaction instructions. The given document will not define cases of use of this flag. The value TRT must be zero.

Reserve

Must be set to 0.

TIME_TR

1 octet. Time of transaction life in 2 - second intervals (maximal lifetime - 8 minutes). The receiving side begins readout of this time after receiving all transaction instructions. The value %x00 sets transaction without restriction of lifetime.

In the last instruction of transaction the header, _END_CHAIN is always sent.

7.2.2 EXEC_TR

This instruction "To execute the transaction" (EXEC_TR) is transferred for execution transaction early transferred. It has the following values of fields:

```
OPCODE = 158
ASK = 1
PCK = %b01/10/11
CHN = 1
EXT = 0/1
CHAIN_NUMBER - Contains the number of chain, which is necessary to
               execute.
INSTR_NUMBER = 0
OPR_LENGTH = 0
```

7.2.3 CANCEL_TR

The instruction "To cancel transaction" (CANCEL_TR) is transmitted for a cancellation of execution transaction transmitted before. It has the following values of fields:

```
OPCODE = 159
ASK = 0
PCK = %b01/10/11
CHN = 1
EXT = 0/1
CHAIN_NUMBER - Contains the number of chain, which is necessary to
               cancel.
INSTR_NUMBER = 0
OPR_LENGTH = 0
```

The instructions, of which the cancelled transaction consists, delete without a possibility of restoration.

7.3 Fragmented instruction

UMSP is designed for work with the transport protocol with the limited size of transmitted data segment. The fragmentation of the instructions is made in the following two cases:

(1) If the instruction is longer than the maximal segment size of transport layer or,
(2) If the segment is formed of the several instructions and last instruction is not located in it completely.

The decision on fragmentation is taken to UMSP level.

The fragmented instruction is encapsulated in several NOP instructions. Then all instructions NOP are transmitted, as one chain of special type. The following algorithm is used during encapsulation:

(1) The fields SESSION_ID and REQ_ID from the fragmented instruction are written in the first NOP instruction. If field REQ_ID is not present in the initial instruction, it must not be in the NOP instruction. The field SESSION_ID always is present in the fragmented instructions.
(2) Then these fields delete from the initial instruction. The value of all other fields of the header does not change.
(3) After that, the initial instruction is divided into fragments of necessary length. Each fragment is located in a field of operands of the NOP instruction. Other data should not be entered in operand field.

_BEGIN_FRG. The extension header "The first fragment" is transmitted to the NOP instruction, which contains the first fragment. It has the following values of fields:

```
HEAD_CODE = 5
HEAD_LENGTH = 0/2 ; Depends on subordination of the chain.
HOB = 1
Data:
   2 octets: Number of the parental chain. Fragmented instruction
             may be a part of the sequence or transaction.
   2 octets: The instruction number in the parental chain.
```

The header _END_CHAIN is transmitted in NOP instruction, which contains last fragment.

7.4 Buffering

In the given item, the buffering used by the protocol on receiving of data is described. The question of buffering on sending lies beyond the scope of the protocol.

If the instruction is not include in a chain, it is transmitted to VM for execution at once and does not require buffering at the protocol level. The interface UMSP - VM must provide asynchronous instructions sending. It is recommended, that the productivity of UMSP systems, should allow to process the instructions accepted from network, with that speed, with what they were received. All instructions are designed so that carries out the known and limited computing loading. Exception is the instruction of control transfers, which must be processed in two stages. The instruction correctness is checked firstly and its scheduling is made. Then the instruction is executed. At that must be guaranteed that the protocol can receive such part of processor time, which would allow it to work in stationary mode. Therefore, the questions of node overload are deduced on VM layer and user applications layer, where they can be sensible controlled.

For chains, the protocol provides two schemes of buffering during the receiving:

(1) At the session connection establishment, the sides agree about the allocated buffer ("window") size. The window always is more than the maximal segment of a transport layer. The transmitting side can expect for this buffer without the preliminary coordination with the receiving side. The window size is established single for each session connection, and cannot be changed in subsequent. UMSP is designed for using of transport layer, which informs about the data delivery. Therefore transmitting side traces the current free size of the window on the reception side for each connection without assistance. If the reception side finds out, that the data have been received, which cannot be placed in the window, the connection is broken off.

(2) For transactions and fragmented instructions, which size exceeds the window, it is necessary to request the reception node the sanctions to sending. The theoretical limiting size of chain transmitting so is 4 Gbytes.

REQ_BUF. The instruction "To request the buffer" requests at VM the buffer allocation for sending of transaction or large fragmented instruction ("Window"). It has the following values of fields:

```
OPCODE = 24
ASK = 1
PCK = b01/11
CHN = 0
EXT = 0/1
OPR_LENGTH = 1
Operands:
   4 octets: The buffer required size in octets. The value is
             equal to the total size of all instructions of the
             chain, including the size of the subordinated chains.
```

The instruction is formed under the initiative of the protocol and it uses the instruction RSP_P as acknowledgement. However, on the reception side the buffer is allocated at VM level, as VM has the most complete information about the task. The interface between UMSP and VM must give possibility of asynchronous request of such buffer.

The instruction REQ_BUF can be used irrespective of the possibility to place the chain in the buffer, allocated for session (window). It is necessary to take into account, that the negative acknowledgement can be transmitted on this instruction, but using of a "window" guarantees sending.

The subordinated chain on reception uses the buffer of the parental chain.

The sequence sending will not require about the buffer allocation in difference of transaction or fragmented instruction. If the single connection TCP is used for sending, the sequence buffering is not necessary. If the multiple connections TCP with multiplexing are used, the sequence requires buffering for the disorder instructions. In this case, it is necessary to use the buffer, allocated for session.

Transactions, at which flag TRR = 0, always must request the sanction for sending by instruction REQ_BUF, even if they can be placed in one segment of transport layer.

The buffering of the fragmented instructions and transactions, at which flag TRR = 1, depends on their size:

- If the transaction is located in one segment of transport layer, it is transmitted without buffering.
- If length of a chain is no more then "window", it can be transmitted without request of the buffer of window allocation. Thus, the place in the buffer must be reserved before the sending begins. The sending cannot be begun, if it is not enough places in the buffer. In this case, it is possible to wait the window deallocation or to use the request instruction of the buffer allocation at VM REQ_BUF.
- If length exceeds the session window size it is necessary to use the instruction REQ_BUF.

7.5 Acknowledgement of chains

The field REQ_ID in chains of any type is established only in the first instruction and concerns to all chain. The all following instructions, including last, do not contain REQ_ID.

The transport protocol used for chains sending, must inform about the end of data transfer, because it is necessary for the transmitting side to know the free size of the allocated session window on the reception side.

If the chain uses the allocated VM buffer (the sanction to sending REQ_BUF was requested), or the chain completely locates in transport layer segment, the protocol on the transmitting side does not trace acknowledgement.

If the sequence is transmitted, the transmitting side receives the information about free place of the buffer on the reception side by acknowledgement of transport layer delivery. It can be made, as the regulated sequence instructions are transmitted VM at once after receiving and release the buffer.

The fragmented instructions and transactions are not transmitted VM until its will be completely accepted. If session window is use, the occupation of places in the buffer can be calculated upon acknowledgement of transport layer sending. To trace free of places it is necessary to check execution acknowledgement by VM. The following algorithm of sending is used for this purpose:

- The value of field REQ_ID, which has given VM for chain sending, is kept and it is enters the value established by the protocol instead of it
- The new value REQ_ID is transmitted in the first instruction of chain
- The chain completely collected in the session window on the reception side. After linking, it is transmitted for execution on VM. At that, the chain can continue to occupy a place in the buffer.
- After execution, VM informs about it to the reception side protocol.
- The protocol clears place in the allocated buffer.
- Then the protocol forms and transmits on chain acknowledgement RSP_P, instead of RSP, as in other cases.
- The transmitting side protocol corrects size of free place in the reception side buffer after reception of acknowledgement RSP_P.
- Then the old value REQ_ID is restored and the acknowledgement is transmitted to VM.

7.6 Base-displacement Addressing

The memory base address for the relative addressing can be established for the instructions from one chain. Thus, it is possible to use the abbreviated address memory fields in the instructions of chain. The abbreviated addresses are used, as displacement from base.

_SET_MBASE. The extension header "To set memory base" establishes the value of base address for chain. It has the following values of fields:

```
HEAD_CODE = 7
HEAD_LENGTH = 2/4/8   ; Depends on address length.
HOB = 1
DATA contains:
```

4/8/16 octets: The base address.

The length of address is 3 octets, enters the name in last octets of 4-octets data field. The initial octet is set to 0. The base-displacement addressing is not used for nodes with address length 2 octets.

The value of memory base for a sequence may change. The base must be established once in any instruction for all transaction instructions. The repeated establishment of transaction base is a mistake, which results refusal of transaction execution.

8 Extension Headers

This section contains the description of the extension headers, which are not connected with the definite instruction. The description of the specialized extension headers describes in the appropriate sections of this document.

8.1 _ALIGNMENT

The extension header "Alignment" (_ALIGNMENT) allows to make any extension header or field of operands multiple of 4 - 16 octets with the step of two octets. The protocol does not give any rules of use given extension header. It can be used arbitrarily. The header has the following values of fields:

```
HEAD_CODE = 8
HEAD_LENGTH = 1-7 ; Depends on length of the data field.
HOB = 0
DATA contains:
   2 - 14 octets: All octets of the field have the zero-value.
```

The format of the protocol instructions provides the alignment of two octets field without any additional means.

8.2 _MSG

The extension header "The any message" (_MSG) allows sending the textual message in symbols ASCII. The order of this header processing at receiving can be anyone. The message can be written in a log-file, be shown on the console or be ignored. The header has the following values of fields:

```
HEAD_CODE = 9
HEAD_LENGTH = 1 - 127 ; Depends on data length of field.
HOB = 0
DATA contains:
```

2 - 254 octets: The any text of the message.

The instruction may contain several headings _MSG.

8.3 _NAME

The extension header "The Name" (_NAME) allows specifying the job name, name of object or name of object procedure. The header has the following values of fields:

```
HEAD_CODE = 10
HEAD_LENGTH = 1 - 127 ; Depends on length of a field of data.
HOB = 0
DATA contains:
   2 - 254 octets: The text of the name in symbols ASCII.
```

8.4 _DATA

The extension header "The Data" (_DATA) is used for data transfer in the instructions of exchange between VM, if the data cannot be placed in operands. It allows transferring up to 4 Gbytes of data in one instruction. The header has the following values of fields:

```
HEAD_CODE = 11
HEAD_LENGTH = 1 - 2 147 483 647 ; Depends on length of the data
                                  field.
HOB = 1
DATA contains:
   2 - 4 294 967 294  octets : Binary data in an any format.
```

8.5 _LIFE_TIME

The extension header "The lifetime" (_LIFE_TIME) contains value of time. It has the following values of fields:

```
HEAD_CODE = 12
HEAD_LENGTH = 1/2; Depending on length of data.
HOB = 1
DATA contains:
   2/4 octets: The time in 1,024 milliseconds intervals.
```

The header _LIFE_TIME allows to set limiting time of sending of the instruction to VM of the addressee.

The instruction lifetime is calculated as follows:

- On the transmitting side the time of waiting in a queue to the transport layer is taken into account. The value of the lifetime decreases on the waiting time value now of the transport layer package formation.
- On the reception side the lifetime is taken into account only for the fragmented instructions. The value of the lifetime decreases on time of the instruction assembly value. This header is ignored at receiving for no-fragmented instructions. Its value must be sent to VM.
- The time of sending at the transport layer is not taken into account. For the fragmented instructions, only the time of sending of the first fragment is not taken into account.

The end of lifetime at the instruction relating to sequence finishes the sequence sending. The header _LIFE_TIME must not be used at transactions sending.

If the instruction is fragmented, the header _LIFE_TIME is sent only in the instruction NOP, containing the first fragment. This header deletes from the initial fragmented instruction. If the time is over, when the fragmented instruction part has not been transmitted yet, the stayed part of the instruction is cleared.

The instruction lifetime is established by the sender VM and must be sent together with data to the addressee VM. If the time of life expires, the instruction is rejected and the negative response (if ASK = 1) is sent to it. If ASK = 0, the response is not sent.

The header _LIFE_TIME may be used in the multimedia systems and in the real time systems. The protocol may raise the priority of sending for data with coming to the end lifetime.

9 Search of resources

Virtual Machines are the identified resources of the protocol. The VM standardization is not function of UMSP. The protocol gives transparent environment for transportation of the code and data of any type.

For VM, connected to the protocol, the following values are established:

- The VM type. The range of values 1 - 65534.
- The VM version. The range of values 1 - 65534.

The protocol requires obligatory compatibility from bottom-up for VM of one type and different numbers of the versions (VM with larger number of version must be able to execute the VM code with any smaller number of version).

Numbers of VM types are broken on the following ranges:

1 - 1023	Assigned for standard VM
1024 - 49151	Assigned for registered VM of the users
49152 - 65534	Free (defined for dynamic and/or private VM)

Numbers of types and versions %x0000 and %xFFFF are reserved by the protocol.

Several VM of different types may be united in a group. All VM, included in a group, must work in the common space of local memory and have the common subsystem of the jobs control. It means, that if the same 128-bit address is met in anyone VM code for one task, it must specify one physical cell of memory. The performance of the specified conditions allows executing multivendor user code (containing procedures for different VM) on one node. All VM, included in a group, must have the different types. The group can include no more than 65534 VM. One number of group on different nodes may identify groups with different structure VM.

To each group VM on the node the code of group of 2 octets length is assigned. So long as the node has even one session connection, the codes of groups must not change. It is recommended to change the code of group only at reconfiguration of the node. The group VM is identified, as well as one VM. Thus, the type VM is set to 0, and the number of group is assigned to VM version.

The support of association VM in groups is optional requirement of the protocol. The multivendor user code can be executed, even if the association in groups is not provided. For this purpose, the procedures containing a different type of a code must be executed on different nodes.

UMSP gives the instructions of search of the VM, which allow defining, what VM and the groups VM are connected at the given moment to the protocol on the definite node.

The instructions of search of the VM can be sent upon TCP or UDP. The broadcasting dispatch can be used. The node can independently notify about VM, available on it, for example at start, or to respond on others VM requests. The answerback instructions must be sent under the same protocol, on which the request was received.

VM from ranges of numbers 49152 - 65534 or any group VM may be identified on names. VM with numbers 1 - 49151 must not have names at a layer of the instructions UMSP.

9.1 VM_REQ

The instruction "To request the VM" (VM_REQ) allows finding out VM, connected on the remote node. The instruction has the following values of fields:

```
OPCODE = 25
PCK = %b00
CHN = 0
ASK = 0/1
EXT = 0/1
OPR_LENGTH = 0 - 65534 ; Depending on quantity VM in operands.
Operands:
   2 octets: The type required VM. The value 0 is not allowed.
   2 octets: The version required VM. The value 0 is not allowed.
             The value %xFFFF requests the most senior version.
   .
   .
   .

   2 octets: The type required VM.
   2 octets: The version required VM.
The optional extension header:
   _NAME - This header contains the name of required VM or VM
           group.
```

The instruction without operands is used for request of all types VM, connected on the node. The instruction with one VM in operands requests the information on one VM. If it is contained several VM in operands, the group VM containing all specified VM is requested. The type and version in list VM must be indexed on increase.

To request VM, used at work without session connection, the VM type and VM version must have the value %xFFFF.

The header _NAME is not connected with value of operands. For it, the separate answer must be transmitted.

9.2 VM_NOTIF

The instruction "To notify about VM" (VM_NOTIF) is used for the notification of one VM or one VM group attached on the node. The instruction has the following values of fields:

```
      OPCODE = 26
      PCK = %b00
      CHN = 0
      ASK = 0/1
      EXT = 0/1
      OPR_LENGTH = 1 - 65534  ; Depending on quantity VM in operands.
      Operands:
         2 octets: The used transport protocol.  The following values of
                   this field are definite:
            x0100 - Single TCP connection through the port 2110.
            x0101 - Multiple TCP connection through the port 2110.
            x0102 - Single TCP connection through ports 2110 and UDP
                    through ports on receiving 2110.
            x0103 - Multiple TCP connection through ports 2110  and UDP
                    through port on receiving 2110.
            The port 2110 must be opened on the one side or both side at
            each TCP connection.
         2 octets: Reserved.  This field must not be analyzed by the
                   protocol during the receiving in the current
                   realization of the protocol.  It must be set to 0 at
                   sending.
         2 octets: The type VM.
         2 octets: The version VM.

         .
         .
         .

         2 octets: The type VM.
         2 octets: The version VM.

      The optional extension header:
         _NAME - This header contains the name by separate VM or group VM
                 from operands of the instruction.
```

It is necessary to generate several instructions, if it is required to inform about several VM or groups. It is necessary to form the separate instructions for each protocol, if the node provides several transport protocols.

If the instruction is used for the response to VM_REQ request, it can contain ASK = 1 and REQ_ID, established in value from the instruction of request. If the VM group was requested, the instruction must contain several VM. First VM must have the type set to 0 and the version must contain the number of group. Others VM must define structure of group. The type and version in VM list must be indexed on increase.

The protocols, contained in the instruction VM_NOTIF, may differ from the protocol, through which this instruction is transferred.

10 Security Considerations

The present document contains the description of the functions, minimally necessary for the realization of the declared task - immediate access to memory of the remote node. To reduce initial complexity of the protocol, the decision of safety questions is not included in the document. All reasons of the given unit are the recommendations to the further expansion of the protocol.

For the description three nodes are used - node A and node B are exchanges the data. The node G is JCP.

Protection against sniffing, spoofing and hijacking:

(1) The means specifies in TCP/IP can be used.
(2) There is a possibility to create chains with the special processing. To create such chain, it is necessary to transfer the extension header, determining the special processing, in the first instruction of the chain. The instructions of chain can be encapsulated in the NOP instructions. The algorithms of the control of instructions sequence integrity or the encryption can be realized in such a way.

Protection against the man-in-the-middle:

The protection is based on the fact, that the routes between nodes A - B, A - G and G - B is not crossed. Such scheme allows organizing the additional managing dataflow, allowing revealing such type of attack. If the specified routes pass through one gateway, this protection is less effective.

Authentication:

The protocol working is based on a principle of the centralized control. It allows using several schemes of authentication. The parameters of authentication are sent in the extension headers. The establishment of session connection can contain up to eight handshakes. It also raises flexibility at a choice of authentication algorithm. The realization of authentication is possible between three pairs nodes A - B, A - G and G - B. All pairs can be used in any combination. The node G can be specially allocated for realization of authentication.

Protection against denial-of-service:

The instructions of the protocol have definite computing loading. It allows projecting the node so, that it can process the instructions with such speed, with what they are accepted from the network. A possible reason of an overload is the instructions JUMP and CALL. VM must solve this problem. It has the complete information about the user task and can make a decision on the amount of allocated resources. The decision of a problem is the failure in service for low-priority traffic.

Protection at the applications architecture level:

The protocol allows creating the applications of any architecture. It is possible due to an asymmetric structure of connection. It is possible to allocate three basic groups:

(1) The client who is carrying out terminal functions and client/server technologies. The security of such systems is completely defined by the server. Such architecture is represented most protected.
(2) The client, loading an active code from the server. It is the least protected architecture, from the client point of view. On the server side, there are no special requirements upon protection.
(3) The client, who is executing his code on the server. This architecture is safe for the client. It is necessary to strengthen the protection on the server. The functionalities of such architecture do not differ from architecture of loading by the client of an active code. If ones take into account, that the server is the specially allocated computer, the given architecture is optimum.

All given technologies may be used simultaneously in any combination.

11 Used Abbreviations

API Application Programming Interface.

CTID JCP assigned the Control Task IDentifier to each task of the job. Its length is equal to length of the local address memory on the node JCP.

GJID Globally Job IDentifier is assigned for the each job. GJID is defined on the JCP node. It has the same format, as the 128 - bit address of node JCP memory has. The address of local memory is replaced on CTID of the first (initial) task of the job in it.

GTID Globally Task IDentifier is assigned to each task. GTID has the same format, as the 128 - bit address of node memory has. The address of local memory is replaced on LTID in it.

JCP Job Control Point. This node will control the job.

LTID Locally Task IDentifier is assigned to each active task on the node. LTID length is equal to the local memory address length defined for the node.

VM Virtual Machine.

12 References

[1] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[3] Crocker, D., and P. Overell. "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[4] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", STD 7, RFC 793, September 1981.

[5] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[6] Srinivasan, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 1831, August 1995.

13 Author's Address

Alexander Y. Bogdanov

NKO "ORS"
22, Smolnaya St.
Moscow, Russia 125445
RU

Phone: +7 901 732 9760
EMail: a_bogdanov@iname.ru

14 Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.